MASTER IN COMPUTER SCIENCE

Crowdsourcing applied to Smartparking : challenges and prototype development

Bogaerts, Gary; Jones, François

Award date:
2017

Awarding institution:
University of Namur

UNIVERSITÉ DE NAMUR
Faculty of Computer Science
Academic Year 2016–2017

Crowdsourcing applied to Smartparking : challenges and prototype development

BOGAERTS Gary JONES François

Supervisor: _____ (Signed for Release Approval - Study Rules art. 40)
Bruno Dumas

Co-supervisor: David Gillot

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

Acknowledgements

We first would like to thank our thesis supervisor Bruno Dumas for his excellent guidance. His door was always open to us and we are grateful for the many insightful recommendations he gave us during the writing of this thesis.

We also would like to express our sincere gratitude to David Gillot, the CTO of CommuniThings, for his help on theoretical, technical and logistical aspects, and also for all the resources he provided us with.

We would also like to thank all members of CommuniThings that have helped us along the way, P. Rodríguez, J. Tomero, X. Sand and L. Schmidt for sharing with us their experience as software developers.

Finally we also would like to thank our families, friends and neighbors, for their constant love and support.

Contents

Introduction

"What is the city but the people"
William Shakespeare - The Tragedy of Coriolanus

"Please let there be a parking spot nearby". A little prayer we all made at least once in our life as drivers, and certainly more than once when driving in urban areas. Car parking in cities has always been a nightmare, and the increasing amount of vehicles in use is making the problem worse everyday. But if there is something mankind can often relies on to make daily life easier, it is the increasing usage of technology, particularly automation.

With the emergence of the Internet of Things since the early 2000, the huge increase in the amount of smartphones owners, the improvement of the computational power of those phones and associated sensors, it is then not surprising that the scientific community has been searching for technological solutions for common issues in cities such as parking. Improving cities as a whole through the usage of technology is the main ambition of what we call nowadays "Smartcity" projects. With Smartcities naturally came the term "SmartParking", a notion that encompasses many solutions to the parking issues in urban areas, taking very different forms and integrating numerous IoT concepts.

In parallel, Crowdsourcing is also a domain related to information technologies that has considerably grown in the recent years. With the increasing amount of data and bandwidth handled by computers and IT systems, a large amount of information is made available and this information may become more useful everyday. With this in mind, asking people to provide the information can prove to be a relatively cheap approach to many challenges, which is the definition of Crowdsourcing.

Interestingly, Smartcities and Crowdsourcing have grown alongside one another for years now, and more than once they have been merged with the objective of creating solutions improving peoples daily life. But in the more specific case of Smartparking, the synergy of the two haven't been applied deeply yet.

This is why we wanted to explore this idea with our master thesis. Collaborating with CommuniThings, a company already active in the SmartParking business, but less in the crowdsourcing area, gave us insight of the current state of the field. But since we wanted crowdsourcing to be an important part of our thesis, the idea came to try to develop a SmartParking solution for CommuniThings that would massively rely on crowdsourced data. But without prior extensive knowledge of the crowdsourcing domain, it became important to identify the biggest impact crowdsourcing could have on smartparking solutions. Hence our research question :

"What are the major themes to consider when relying on crowdsourcing for a Smartparking system"

Smartparking systems can take many shapes and integrate very different concepts from one another, even without taking crowdsourcing into account. Our objective was then to determine which of those concepts were less useful when relying on crowdsourcing, and which were even more important, in order to then be able to concentrate our researches on those concepts and their implications, and finally find ways to use them to develop a prototype that would efficiently apply Crowdsourcing aspects to Smartparking.

Methodology

First thing to consider is that the choice to do a single master thesis in duo was a choice we made and was not something we were stuck with by having a similar subject. Gary and I have been collaborating in nearly all our academic group projects and reports (meaning all of them when groups weren't randomized) for the last 4 years, and working alongside one another as become really natural for us, and has proven time and time again to be more effective for the both of us.

Thus when time came to choose a master thesis subject, doing two master's thesis or one combined was never really a question as long as it was authorized by the university, which obviously was. We then had to find a subject that would keep us both interested and kindle the interest of one of our professors to accompany us in this journey. Through contacts with CommuniThings, for whom Gary had worked previously, and already done his bachelor graduate thesis at the Institut Paul Lambin (from where we both graduated before coming in Namur for our Master), we established a global theme around crowdsourced parking, a subject that happened to share common points of interest with a subject that Professor Dumas was proposing, and after a meeting with him and David Gillot, CommuniThings CTO, both agreed to supervize us for this thesis.

It was decided early that our thesis would have a theoretical approach, a state of the art related to SmartParking, its objectives and impacts, as well as the crowdsourcing advantages for smartparking, and a practical approach, a proof of concept of what we researched, and something that could merge with the others projects CommuniThings was working on. Those two approaches are represented by the two parts of this document.

For the theoretical approach, our methodology of work as a duo was first to identify SmartParking solutions developed in the recent years, including or not crowdsourced component, and then to determine the limits and issues of these solutions. Afterward, and in collaboration with CommuniThings, we started to imagine what our ideal system would look like, and the concepts behind it. The importants concepts we wanted to try out led us to determine several main themes of research. Those theme would be our biggest challenges to realise our SmartParking solution, and also compose our theoretical answer to our research's question, the application of crowdsourced concepts to the smartparking domain.

Having our domains of research identified, we spitted up the work depending on who was most interested by each theme. Gary being way more obsess about security and privacy issues than myself (sometimes to an annoying extend communication-wise), he took over those chapters, while being the more social person, I was more interested in the community development and users motivations to participate to crowdsourced platforms, and also in the data reliability since it was in a way related to the community in the sense of "How can me trust our users". The remaining chapter we wanted to develop, about the Economic design of our application was done in collaboration by having both of us find information on other applications economic models and then compare them.

As for the practical work, our task was divided in several milestones. The first one was to develop a small application to familiarize ourselves with the technologies used by CommuniThings, and having as a purpose to register multiple sensors values such as gyroscope and accelerometer, in save those value in CSV files that were afterwards used by CommuniThings to generate a tree of decision able to determine if a user is walking, driving or standing still based on his smartphone sensors. This was also the first step in what was going to become the passive parking detection.

For that application, our tasks repartition was to have Gary work on the sensor's data collection, while I was developing the UI and binding it to the processes Gary was implementing. All of this was done with NativeScript.

Once this task was accomplished, we were finally able to dive in the main application itself, the SmartParking system. For that milestone, we had an enormous system to

develop. On one side, the mobile application, with all its UI and inner mechanisms, and on the other side, the server handling the requests and trades. Since these two systems had to be done in parallel, it was rapidly clear that our best course of action was to have one of us working on the mobile application and the other on the server. Thus it was decided that I would be doing the mobile part, while Gary was working on the server side of things.

For both the server and the mobile application, we were given access to CommuniThings GIT repositories, and created a branch of their existing systems. This meant that we were able, thanks to CommuniThings, to rely on the work previously done by their developer as a base for our own tasks. For example, the map module of the mobile application was already existing in their own application, and we "simply" had to modify it and the interface to add our own business logic. Additionally, this also meant that CommuniThings's developer where able to guide us with the existing systems, as they were obviously familiar, much more than we were back then, with the frameworks.

Even if we were working on systems being on appearance independent, it was necessary for us to work together, in the same room. Firstly because our system had to communicate and the chances of our communication messages working as intended on first try was really low, and we had to make adjustments continuously to either system, which is obviously easier when we both work in the same room. Secondly, because even if we were working on different systems, having the other part of our duo close by was often a big help when stuck on some bugs or unexpected behaviors.

Within our system as a whole, we also implemented features step by step, reaching milestones one after each other. We first developed our users related functionalities (user registration, login) as none of those were existing within the CommuniThings systems. The following steps were the management of the vehicle, related to the users. This included all the features to allow an user to register its vehicle, to the management of vehicle's conflict of ownership.

Once we had users and vehicle all the way down to the database, we were able to start working on the trades themselves. The firsts iterations were really basic, and we progressively added more feature to the trades, ending with the handling of the points used for the trades.

Globally, things went rather smoothly, as we were used to work together, and known each other for quite some time now, which facilitates our interactions when we know the work ethic and habits of the other.

Part I

Smartparking and Crowdsourcing

Concepts and Related Research

Chapter 1

Related Works & Objectives

1.1 Overview

The Internet of Thing is a term first used in 1999 by a British researcher named Kevin Ashton. Even though some of the core concept pre-date 1999, the name remained and since then it has become a constantly increasing field of research and a bigger market every passing year.

Although it comprise many concepts and systems, the IoT field can be described as the inter-connection between "Things", that could either be (but not limited to), common everyday life items, buildings and vehicles. The specificity of the IoT is to complement these items with sensors, controllers and other led, alongside an Internet connection, usually through WIFI. The whole item is then connected to others system such as smartphone for computation of the data, and to servers for the storage [10].

One of the most promising uses for emerging "Internet of Things" technologies is improvement in urban mobility.

Currently, a typical commuter car is parked on average 95% of the time [32]. This represents a huge burden on infrastructure as an important surface needs to be dedicated to the task of hosting parked cars. On top of this waste of space, traffic management in metropolitan environments is also a critical aspect of urban planning where the commute model of today is experiencing limitations, especially when the flow of vehicles is exacerbated during peak hours. For example, it is estimated that around 30% of vehicles on the road in city centers are searching for a parking spot [2]. This in turn has measurable consequences in terms of air pollution.

Self-Driving vehicles could potentially solve these problems almost entirely by reducing the total number of vehicles in circulation, as shared self-driving cars can be much more often on the road providing a useful transportation service. However, self-driving vehicles are not yet widely accepted and extensive investment in infrastructure and manufacturing would probably be needed before mass adoption is possible.

Another possible solution to the traffic congestion issue is an ensemble of applications, system and other prototype sharing the name SmartParking. The concept isn't new and has been around nearly since the beginning of the IoT. A lot of Smartparking solutions have been developed in the past years. They share in common a will to reduce traffic congestion in urban centers through a better organization of the parking lot and parking spots attribution.

1.2 SmartParking

First of all, it is important to define clearly what SmartParking is, as the term has been used a lot and in multiple contexts. SmartParking systems can be very different from one to another, as sometimes, even though the final goal of the system remains an automation or better organization of the parking spots, the purpose of the systems differs greatly. To get a better grasp of the differences between such systems, a study [14] has sorted SmartParking solutions in five categories.

1.2.1 Parking Guidance and Information System (PGIS)

The main category, also the one that usually comes in mind first when talking about SmartParking. The goal of PGI systems is to provide assistance to users for parking by giving indications on the occupancy of areas that can either be cities or defined parking lots. Such systems usually relies on vehicle detection sensors that detect when vehicles are leaving or arriving to parking spot, in order to obtain a global view of the occupancy situation. This information is then transmitted to the users and the system can then redirect them to the areas with the lowest occupancy, improving their chances to find a spot.

1.2.2 Transit-Based Information systems

Similar to PGI systems, Transit-based systems also focuses on directing users toward areas where they will find parking spot, but those systems emphasize on park-and-ride facilities, and also provide real-time informations about public transportations schedule, availability and traffic conditions. Those systems are designed to have the user plan its route in advance by knowing which means of transportation he will be able to use to optimize his travel time.

1.2.3 Smart Payment systems

These systems aims to either improve or replace existing parking payment methods, such as parkmeters, that can still be found in nearly all cities. The objective behind new and technology assisted payment methods is to reduce the maintenance cost of the city infrastructure dedicated to parking management, and also reduce the dependance to cash money in parking situations.

1.2.4 E-Parking systems

These systems offers the possibility to reserve parking spots within defined parking lot, to ensure the users that they will find a spot in the parking area of their choice, with absolute certitude, something that PGIS and TBIS cannot really offer. Such systems are sometimes coupled to PGIS.

1.2.5 Automated Parking

This category refers to parking lots that attribute spots and place the vehicle in it automatically through computer controlled mechanism. The purpose is the optimization of the parking space. Example of such system : FATA Skyparks[1]

Going further

In the case of this master thesis, the most interesting systems, in conjunction with CommuniThings needs, are the PGI systems. Smart Payment and Automated Parking are systems rather specific and are not really what we aimed to explore within the SmartParking field. E-Parking solutions has some interesting concepts, the reservation of parking spots, but mostly when coupled to other systems and are not only relying on defined parking lots. TBI and PGI systems share common patterns, purposes and requirements to work, and are what is commonly referred to when talking about SmartParking. We thus have oriented our research toward such systems to discover what was already existing as SmartParking systems providing assistance to find spots in urban area, which is one of the work field for CommuniThings. In the following section we will present some solutions to the SmartParking question, including CommuniThings very own “StopBuy”.

[1] http://www.fatainc.com/product_automated_parking_systems.htm

1.3 SmartParking applications

1.3.1 CommuniThings's StopBuy

CommuniThings[2] is a startup company founded in 2014 in Brussels. The company is active in the Internet of Things and is involved with multiple SmartCity projects. The company's aim is to bring value to citizens, businesses and authorities alike, by providing and combining multiple services.

CommuniThings leads "Sense and the City", a pilot project in Air Quality monitoring in the city of Brussels. Frequent and geographically distributed measurement of various air pollutants are taken from a mobile platform installed on city vehicles. Health and mobility recommendations are formulated based on the data. Citizens can explore a real-time map of polluted zones.

CommuniThings also offers "StopBuy", a smart parking solution that is targeted at local and small businesses. Occupancy sensors in parking spots detect the presence of a vehicle. Users are allowed to park free of charge for a limited time in the course to increase vehicle rotation. The system reports to city agents when a user overruns the allowed parking time.

The main purpose of our collaboration with CommuniThings is to leverage their experience with Smart City applications in the aim to design a practical model for our "Parksharing" application. Through the "StopBuy" platform, CommuniThings has gained important knowledge on practical considerations when dealing with vehicles and parking spots that we are eager to learn.

Moreover, this project is a great opportunity for us to work with a team of professional developers and integrate a real-world development cycle. Ultimately, our hope is that CommuniThings will be able to use the proof of concept we developed as a basis for a future version of the "StopBuy" platform.

[2] http://www.communithings.com/

The StopBuy Platform

The platform was designed with the specific goal of enhancing the rotation rate in parking spots situated in crowded city centers. A slow rotation rate entails that local businesses are penalized since customers are discouraged by costly or scarce parking access and will tend to shop in larger retail stores with dedicated parking space.

To counter this, users of the StopBuy platform can see in real-time free StopBuy parking spots and their location. These spots allows free of charge short term parking, typically 20 to 40 minutes. Users are requested to choose a convenient free spot and are then guided to it. Business partners can then offer more parking time when the user spends money in their store through the StopBuy application. This combination of incentive is proving to be very effective in the city of Mons where 110 StopBuy spots are currently deployed over 21 distinct zones. While the rotation rate of those parking spots was not quantified before the deployment of StobBuy, it was estimated, based on observation on other parking spots, that only 1 or 2 cars occupied the spot per day (9h-18h). Now, the rotation rate is on average between 8 and 16 cars per day, depending on the street. Communithings has consequently received positive feedback from local business partners that saw a marked increase in visitation.

The StopBuy platform also offers advantages for parking enforcement officers as cars overrunning their allowed time are directly notified to officers through a specialized module in the mobile application. This represents a clear efficiency gain since officers no longer need to visually inspect numerous parking tickets and cars for overtime detection. With the StopBuy platform, overtimed parking have virtually no chances to go unnoticed.

Another module provides statistics about parking spot usage over time, in the aim to support future urban planning decisions with quantified data. The application can also be used to make special reservation for disabled people or deliveries.

As of today, the StopBuy platform works only with dedicated parking spots fitted with specialized underground sensors. The sensors are the main providers of data to the

platform, detecting when cars arrives on the spot and when they leave, as well as the duration of their stay.

While the platform provides measurable benefits in favoring small business and parking rotation, its reliance on sensors represent a cost in infrastructure that would be difficult to scale to an entire city. Moreover, the platform is specifically targeted at local shoppers but other profiles such as commuters will find little value in using this platform.

In order to have further impact on urban mobility while leveraging the existing platform, CommuniThings has expressed interest in developing a park sharing service usable outside of dedicated StopBuy zones.

1.3.2 SPARK: Smart PARKing management system

Intelligent parking management systems are widely in use today for medium and large scale indoor parking. Such systems vary greatly in their capabilities which might include the display of the number of available parking spaces at the entrance [3], the use of colored light indicators to make it visually easier to identify a free spot [4], mobile payment facilities via smartphones [5] etc...

A modern parking management system named SPARK is presented in [33]. The paper presents a feature-rich prototype that makes use of *Wireless Sensor Networks* or WSN in order to reduce costs and facilitate deployment. A dedicated sensor needs to be installed on every parking spot, however sensors are battery-powered and wireless making them autonomous, so there is little deployment cost compared to wired sensors. Sensors used in the SPARK prototype continuously monitoring a single parking spot by infrared light and include colored LED indicators. Sensors detects when a car enters or leaves the spot and generate events that are transmitted to a nearby gateway wirelessly. Sensors also transmit health messages at regular interval. The gateway collects events from a group of sensor and forwards them to the parking management subsystem via Internet.

The parking management subsystem acts as the heart of SPARK. This component maintains a real-time representation of the entire parking space base on the events received from the sensors. Information about the general availability of parking can then be displayed at the entrance via digital signage, but SPARK also includes *Automated Guidance* in the form of small screens installed at every turn inside the parking area. These screens display the number of available spots on their right, left and ahead based on the

[3] http://www.trafficparking.com.au/vehicle-guidance-systems-signs.php
[4] http://indectusa.com/single-space-sensors/
[5] http://www.parkmobile.co.uk/how_pay_by_phone_parking_works

information received from the parking management subsystem, also wirelessly.

The main innovation of SPARK is a reservation system which allows clients to reserve a spot that is currently available with a simple SMS. The parking management subsystem answers with either an acknowledgement of the reservation including the expiration time or with a message warning that all spots are currently occupied.

The SPARK system was implemented at prototype scale for 20 parking spots at the Ubiquitous Computing Research Center (UCRC) where the authors work. It showcased a high degree of usability and reliability in test scenarios.

While systems similar to SPARK are promising for dedicated parking areas, the system makes no mention of user authentication and payment facilities. It is unclear how the reservation system could be used efficiently if the system is open to the general public, as in the case of on-street parking. Furthermore, although wireless sensors are certainly cheaper to deploy than wired ones, they still represent a significant investment and will inevitably require attentive maintenance. For these reasons, we argue that systems similar to SPARK are well suited for private parking areas such as employees parking, but would be difficult to apply to public space and even less to on-street parking.

1.3.3 Reservation-Based Smart Parking Systems

Reservation-Based models aim to eliminate the shortcomings of less involved smart parking systems such as PGI. Mainly the fact that sharing of parking information among drivers increases competition between them. Particularly during rush hours, the scarcity of free parking implies that more drivers will go after the same spot, further increasing congestion. This phenomenon is called *multiple-car-chasing-single-space*. [36] implemented a simulator to investigate different parking strategies and their consequences on traffic density. It compares the four following models, in simulated traffic conditions downtown Los Angeles:

- **Blind Search**: is the *de facto* model for street parking where drivers have no information available. Drivers typically start searching nearby their destination and extend the searching area while no free spot is found.
- **Parking Information Sharing**: This strategy gives approximate information of parking availability to drivers, typically attributing a number of free spot to a delimited area or street. Drivers can leverage this information to avoid areas where all parking spots are occupied.

- **Buffered Parking Information Sharing**: This is the same strategy as above with an extra parameter to mitigate the *multiple-car-chasing-single-space* effect. In this strategy, a limited number of free spots is intentionally hidden from drivers, thus creating a "buffer".

- **Reservation Policy**: Here, drivers are allowed to make a reservation for a specific spot in advance. The paper presents an infrastructure-based design for user authentication via smartphones and bluetooth connectivity on the spot, while reservation can be made via the Internet.

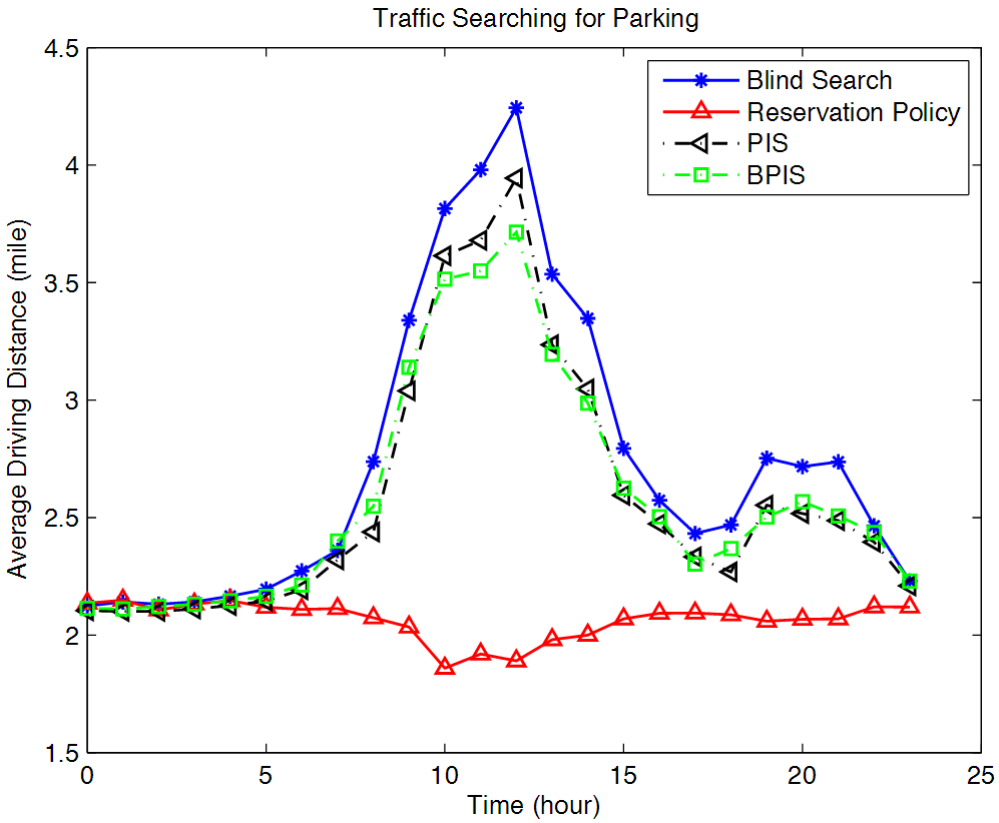


FIGURE 1.1: Distance traveled by drivers searching for parking as a function of the day hour under different strategies [36]

Figure 1.1 shows clearly that reservation policy is vastly superior to the others in minimizing the distance traveled by drivers for parking seeking, especially during peak hours, when it actually decreases. However, this is mainly due to the fact that, as the reservation rate for spots in the city center rises, other drivers are forced to reserve spots that are further away from their destination. This in turn logically reduces the driving distance as drivers go directly to the spot they reserved but increases the walking distance. In short, Figure 1.1 shows that a reservation policy can significantly reduce traffic congestion but not necessarily user satisfaction, as users find parking more easily but farther away.

Furthermore, the reservation model proposed in [36] does not care for malicious users and actually offers little guarantee that the reserved spot will actually be available when the driver gets there. It is unclear, for example, how the system would deter an outsider to occupy a reserved spot, either knowingly or unknowingly.

The reservation policy described in [36] allow drivers to choose which spot they reserve, this implies that the parking information is shared between drivers, at least before reservation, even if a buffering is applied. But it is also feasible to completely automate the process of reservation as suggested in [12], thus eliminating the *multiple-car-chasing-single-space* phenomenon entirely. They present a smart parking system where the parking information is not directly available to drivers. Instead, drivers send a *Parking Request* to the system that includes constraints on cost, location and vehicle size. The system collects the requests from all drivers and periodically performs a group allocation of parking spots that maximizes benefits for all drivers and parking service providers involved.

The system suggested in [12] rely on dedicated sensors for the automatic detection of parking space as well as status indication (i.e. free or reserved) in a manner similar to SPARK [33]. Drivers interact with the system through a mobile application to send parking request and receive guidance to their reserved spot.

The model describes 3 possible results to a a parking request. First, if the system, after an allocation period, fails to find a spot the driver is notified of the failure and is provided with a reason (constraints too tight, no parking available...). Second, if a parking space is allocated but the driver is not satisfied with it, he can reject the allocation and wait for the next allocation period, without guarantee to get a spot again. Third, if the allocated space is convenient for the driver, the spot is reserved and the application provides directions. The driver may be notified on the way if a more convenient spot becomes available.

The authors of [12] recognize the need for a strong guarantee that the reserved spot will ultimately be available for the driver. They argue that status lights on the senors could provide indication that a car is parked in a reserved spot, prompting its driver to move or alert parking operators to tow the car. For on-street parking, the use of "folding barriers" seems to be the only effective solution, although prohibitively expensive.

Taking the decision of the spot's precise location out of the user's hands offers the opportunity for optimizing the utilization of available parking resources. The method suggested by [12] is based on solving a *Mixed-Integer Linear Programing* problem at each allocation period. Pricing is dynamically adjusted based on offer and demand. The authors implemented a simulator in order to quantify the improvements of their smart parking approach over other approaches including *Blind Search* (denoted NG for Non Guided) and *Parking Information Sharing* (denoted G for Guided). The simulation scope

was limited to the campus of Boston University and included 679 on-street parking spots as well as 1932 off-street ones.

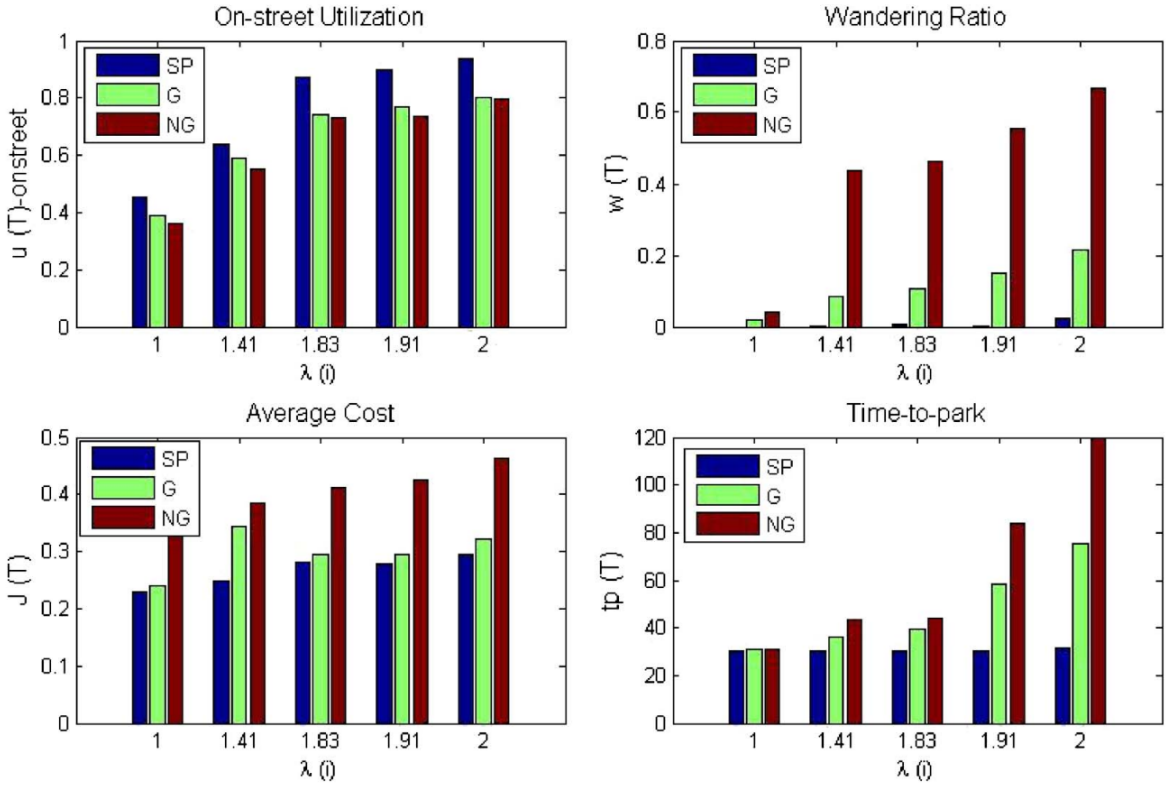


FIGURE 1.2: Performance indicators of three different strategies as a function of the density of parking requests [12]

Figure 1.2 compares different performance indicators as a function of $\lambda(i)$, the parameter of a Poisson distribution of parking requests to a destination i. $u(t)$ is the ratio of utilization of on-street parking spots (occupied spots). $w(T)$ is the ratio of drivers who have reached their destination and are actively searching for a spot because they they didn't get a reservation. $J(T)$ is the average cost of a parking allocation to a user, which includes monetary costs as well as distance, and which the system tries to minimize. Finally, $tp(T)$ is the average *time to park* which is the amount of time between the parking request is sent and the moment the user occupies a spot (for G and NG, this period starts when the user reaches his destination). The results demonstrate better performance on all indicators for their approach to allocation based smart parking.

As is for the system presented in [36], the most important problem in making reservation-based parking sustainable for open-street parking is the lack of guarantee in the availability of the reserved parking spot. In the context of public parking, it would be unwise to rely exclusively on the general public respecting the necessary guidelines (i.e. to not park on a reserved spot) as penalties would be difficult to enforce. This is one of the problem we intend to solve with a crowdsourced approach to smart parking.

1.3.4 Google's Open Spot

Google also experimented with SmartParking. A solution they developed, named Open Spot [30] [16], differentiate itself from other we presented however, as it does not relies on sensors to determine the occupancy of parking spot, but rather on something of major importance for us : crowdsourcing.

Open Spot idea is to let user signal when they leave their parking spot, and then display the information on the map with an icon locating the free spot, and a color associated, indicating how long the spot has been freed. Spots remains on the map for 20 minutes, after what the icon is removed. But 20 minutes is a very long time for parking spot in urban areas to remain free, thus the system have a high risk of mismatching information which will at the end frustrate users

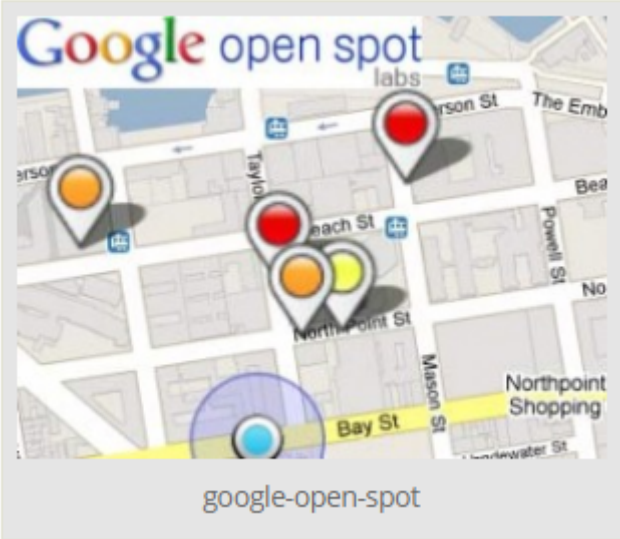


FIGURE 1.3: Open Spot UI [30]

Open Spot possess some ideas that makes it interesting as a starting point for the development of a crowdsourced smartparking system, but in itself the system lacks incentive for users to submit information about their parking spot usage. Users receives Karma points for liberating a spot, but these points do not provide any advantage to obtain a parking spot later, and are thus basically worthless. This led to very few user sending data, which is critical for crowdsourced application where amount and quality of data is a prime. Due to the drawbacks of this system it never gained popularity.

But this case raises interesting points regarding crowdsourcing, and the importance of incentives to contribute to the application's community. Without incentives, a crowdsourced application is most likely doomed.

1.3.5 ParkSense

As we have seen, relying on sensors to determine the occupancy of a spot is not always an option when it comes to on-street parking, either due to maintenance, reliability or the large amount of sensors required. Crowdsourcing is an option to obtain the parking spot information with less infrastructure, but relying on user behavior is not always an effective option. It is then no surprise that researchers try to develop other methods to obtain the information related to the parking spot availability.

One of the option explored by a team of the University of Cambrige [25] is to use Smartphones data and sensors to detect when an user is leaving a parking spot, thus triggering so-called "unparking event" that can then be bind to geographical position through the GPS and WIFI of the phone.

ParkSense, the name of the application they developped, explored the usage of the GPS, the accelerometer and the WIFI to obtain those "unparking events". As the GPS and accelerometer proved too unprecise to be used on their own, the team developped and system to detect these events using the WIFI data, using particularly set of WIFI access points. Those sets are used for two purpose : to determine the status of the user and to know when an user comes back to his car.

Determining user's status

The idea of ParkSense is to obtain regularly the set of WIFI access points detected by the phone, and compare it with the previous sets, using the Jaccard Index, a method used to determine the similarity between two ensembles. Their research showed significant results that this index can be used to determine if a user is walking or driving, as the index is notably lower while driving, which is logical as the sets of access points variates more frequently when the user moves faster, which happens when the user is driving compared to walking. The definition of "unparking events" could loosely be described as WALK-UNPARK-DRIVE, thus by determining the status of the users, the first step of calculating those events is feasible.

Coming back to a car

Through the usage of the access points sets, the team also developed a mean to determine when an user comes back to his car. By detecting parking events as DRIVE-WALK occurrences, the application stores the set of WIFI access points in memory when the parking event occurred. Afterwards, the current set is periodically compared to the stored set. Should they prove to be similar using the Jaccard Index, it is likely to assume that the user came back to his car. This could also be confirmed with the usage of the GPS data. This information, combined with the detection of the user's status, can then be used to

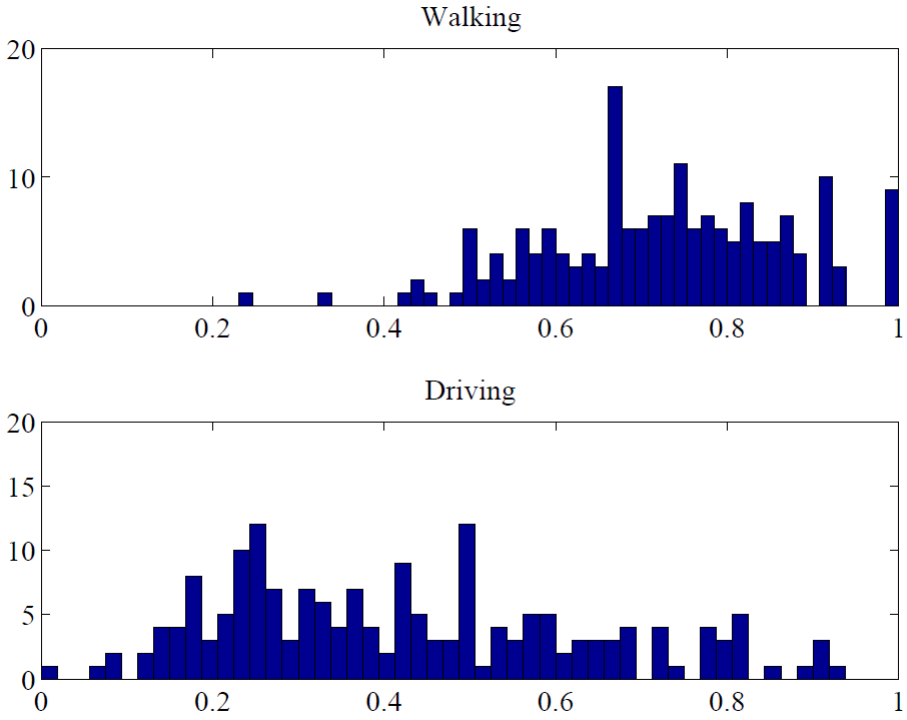


FIGURE 1.4: Distribution of the Jaccard index while walking and driving [25]

determine the "unparking events", which then can be translated to parking availability information.

Issues

With these methods, available spots can be indicated in a map, similarly to what OpenSpot is doing, but do not require any action from the user, which was the main drawback of Google's application. However the system still present some issues, mainly, the event detection is not perfect, and some situations such a parking a car, then getting in another one (i.e. car sharing) can lead to the detection of erroneous events.

The other issue is the time required by the system to determine that the user is actually driving and has left his spot, averaging 5.3 minutes. This means that the spots could only be made available more than 5 minutes after having been freed, leading to a high risk of having the spot being already occupied when someone tries to claim it. Lastly, the energy consumption of the WIFI detection could sometimes increases a lot and become an issue, notably inside buildings.

1.4 Common issues between Smartparking solutions and proposed approach

All the SmartParking systems we presented share a common pattern by design: the user has very little active control in the data collection, with the exception of OpenSpot. Parking spots are usually items automatically monitored through sensors, even if theses sensors are sometimes those of an user's phone. Depending on the solution, the parking areas have to be perfectly defined, such as SPARK parking lots [33], or they could be on-street parking improved with sensors. The solution can be informative only, displaying occupancy, like StopBuy, or allowing user's reservation, the exact reservation procedure depending of the solution [12] [36]. Even if these solutions are feasible, the amount of parking spots accessible through them remains low due to the deployment costs of sensors at large scale.

In order to increase parking rotation rates at larger scale without having to put sensors in every street and every parking area, it is preferable to use existing parking infrastructure, e.g. public parking spots, thus leaving open the detection issue. Google tried to solve this simply by asking people to indicate when they leave a spot, but without real incentive, this is not going to happen. [30], and the automatic detection solution by ParkSense [25] does not provide full satisfaction either.

All these solutions never put platform users in relation with each other directly. The approach we wanted to try in order to improve these SmartParking solutions is a complete, end to end exchange model where users not only signal that they are leaving but actually wait for the other party to arrive before freeing up the spot. While this could be seen as really inconvenient for the leaving user, we argue that the piece of mind offered to the other party, as the parking spot availability is almost guaranteed **for** her/him, greatly counterbalances any annoyance as users would assume both roles alternatively within the community.

Such solution would have the advantage of making every parking spot usable by the system without any modification, resulting in a higher parking availability, but presents several points of concern that the other solutions could usually overlook.

First of all, as we seen with Open Spot, motivating the users to give information on their parking is not easy. Additionally, with our proposed approach, we want them to wait for someone else to park, which is even more annoying. We will have to determine what makes users willing to give informations and participate to a community's life in crowdsourced application, in order to implement a solution that should convince people to be a part of it and be willing to regularly trade spots with others members they might

not even know. This is the focus of the following chapter.

The other points whereas an exchange-based approach distinguish itself from those we presented are the active trade of parking spots, meaning we have to develop an economical ecosystem fair and balanced, and the reliability of the user selling a spot, since if we want to advertise that our system provides a guarantee close to 100% for obtaining a parking spot, we have to make sure that users offering a spot are really offering it and are not some kind of Troll.

Finally, our users will have to be able to identify one to another. For this we will have to ask for users personal vehicle's data, and store it. This raise particular concerns about the security and the privacy implications of our application.

Those four points of concern being the main differences between our approach and the other SmartParking systems presented, we researched in more details existing solutions and options we could use in order to make our application viable.

Chapter 2

Users Motivations and Commitment

2.1 Overview

Crowdsourcing applications and marketplaces for online work face a big challenge when it comes to attractiveness and keeping users motivated to participate. Of course, some rely on financial rewards to gain users, but it's not always an option. A study by N. Kaufmann and T. Schulze [15], proceeded to determine and classify the motivations behind a user participation in a crowdsourcing platform. In this case, they based their study on the Amazon Mechanical Turk.

Their main choice when analyzing the different motivations is to classify them into 2 categories : Intrinsec Motivation and Extrinsec Motivation. Intrinsec motivations are driven by internal rewards such as personal satisfaction, fun and skill learning. On the other hand, extrinsec motivations are thus related to external reward, the most common being money, but also positive feedback on a certain task for example.

The authors then decided to divide the most common motivations into groups. In their study, intrinsec motivations were split in two : enjoyment based or community based. Extrinsec motivations were split into three groups : Immediate Payoffs, Delayed Payoffs and Social Motivation. The motivations they considered and the repartition can be seen in the figure 2.1.

Their study shows that the main motivation on the mechanical turk is without surprise the financial reward. But it's not dominating the motivations by a large margin. Enjoyment based motivations are also highly represented, "Skill variety", the use of a specific skill to solve a certain task, and "Task autonomy", the freedom to be creative in the accomplishment of the task, are approximatively 20% behind the payment, but above all other extrinsec motivations. In fact, only 6 out of the 8 biggest motivations are classified as intrinsec (The only 2 extrinsec being money and what the authors called "Human Capital advancement", basically the opportunity to learn or improve a skill by doing the task).

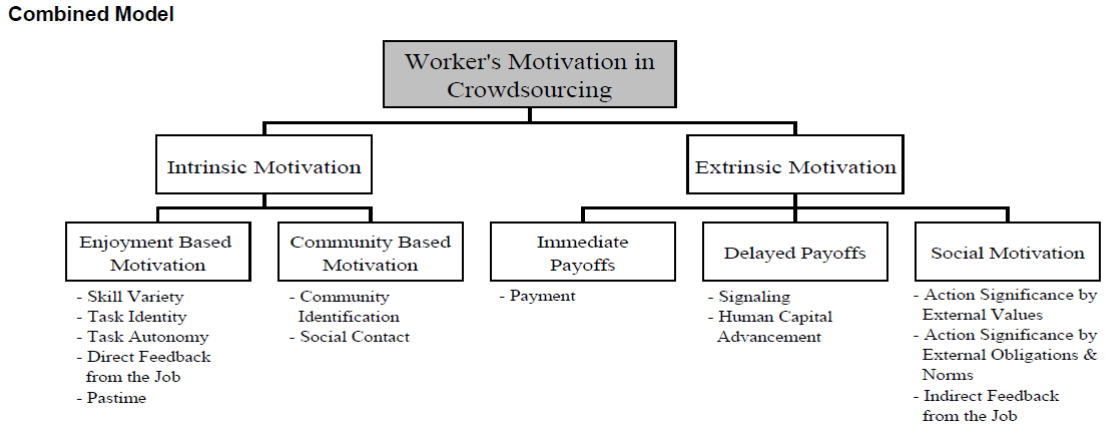


FIGURE 2.1: Model for Worker's Motivation in Crowdsourcing [15]

The study also shows some correlations between the motivations and the time spent on the platforms by a user. The most remarkable relations being the "Community factor", the sense of belonging and "Signaling", the willingness to be noticed by future employers increases the most in importance the more the user spends time on the application. The diversity of the tasks also has an important influence on the motivation expressed by users.

With this study in mind, it will be important to identify the target audience of any crowdsourcing applications in order to adapt the presentation of the information and crowdsourcing objectives to increase the chance of reaching the most out of the participants.

2.2 Analyzing the main factors of contribution

Now that we've identified the main motivation factors for a crowdsourced system, let's break them down a bit and analyze the pro and cons of each one. Indeed if some factors like "Skill variety" or even the financial gain are big factors in general, or when multiples tasks are available, in a system only designed to provide a specific service, like the SmartParking application, those motivators may not apply. So breaking down each one will allow us to determine those we need to focus on the most, since they will interact the best with our application.

Honor to whom it's due : money. It's no surprise that money is the biggest motivator in a crowdsourcing platform, however it's not without consequences. Money does attract people on a platform, because of all the potential rewards. It has an impact on almost everyone, even those looking for fun will not refuse a financial gain, where those looking for it even if the task is not funny and difficult will be doing it specifically for the reward.

But on the other hand, each system using money as a reward is faced with some simple and not so simple economical issues. Simply by considering the supply and demand law, if the task is rather easy, a lot of people will be able to do it and thus it's value will decrease a lot, meaning the potential reward will be very low as well, which in turn, might make people lose interest. Why waste one hour of your time to gain a quarter of a dollar, if it does not provide any fun or learning potential. Furthermore, monetary rewards may also imply legal issues. In some cases, the legality of the transaction can be questioned. In SmartParking, with and without a crowdsourced system, parking spot are the subject of the trades. But those are not the property of the user currently occupying it. It could still be the case, but it is not the primary case we envision for our application. Most of the time people will be parked in public areas, and the rest of the time on some other property. Thus, selling your spot, or even trading with some money involved, implies that you are gaining money over someone else's property, and laws in multiple countries forbid this. For those reasons, using money as an incentive does not seems to be in the best interest for our system.

Self-benefit is a similar motivator to money in such way that the user expects to gain something by using the platform, but he knows that such gain won't be financial and might not even be material. Its attractiveness is less than money since it is not universal and fewer people will be interested. The benefits might not be immediate to the user, but also has advantages over money, since it can provide services that money can't buy, or the promise of a future reward of better quality than an immediate gain. In the case of SmartParking, the self-benefits are a gain of time and gas to arrive to a parking spot, with less stress. Several Smartparking applications presented in the previous chapter had their reservation algorithm trying to optimize the time to park, meaning less gas and more time gained to the users. Self-Benefit is thus obviously an important factor of participation in parking applications, but also works well in conjunction with community driven motivators. For example participation in a Wiki : improving the Wiki quality with personal knowledge and expecting other people will follow your example and improve pages of others, giving you the opportunity to obtain more information on certain subject that you don't know about.

This leads us to another incentive : participation. The study by N. Kaufmann and T. Schulze [15] showed us then the community feeling of a platform, the sense of belonging to a group a people, is a big motivational factor. According to P. Organisciak : "In the realm of crowdsourcing, communities working on the same thing create emotional bonds, whether of respect or hate" [26]. This is the pro as well as the cons of this factor. On one hand, allowing people to interact with others and improving the community by sharing informations procure a sense of belonging to the users which in turn make them feel important inside the community, thus getting them to stay within the community and keep using the system, and make them willing to participate even more, or make other

people (friends, family, co-workers, etc) join the community. On the other hand, a negative interaction for example with an disrespectful user, or bad information from Internet Trolls or simply a computation error can lead users to abandon the platform quickly since they won't be willing to experience this unpleasant interaction multiple times. In such a system, "bad users" must be controlled very quickly to prevent them from doing any long-term harm to the community.

Of course we also have to explore what is basically one of the most universal incentive to do any task : fun. Multiple systems have proposed some sort of "game" to attract users and by making them play, using the data of the game to accomplish some task, sometimes hidden to the players. An exemple from the early 2000 is the ESP Game designed by Luis von Ahn [35]. The principle is easy : two players are shown an identical image, and both have to enter words to describe the image (also known as label), if they use the same label for a picture, they win a point and proceed to the next image. The goal is to reach 15 points in two and a half minutes. By playing the game, you also provide the system with identifiers for each picture, which in turn serve to improve the image recognition task, which by 2005 was considered a really hard task to be done automatically by a computer. (It still is but computers have vastly improved in that regard, thanks in part to the massive availability of images labeled in this way). Regarding pros and cons, there is basically no cons to make a crowdsourcing platform fun, it improves the willingness of the users base to participate and attract peoples. But the difficulty is more in making the task fun. Indeed, some tasks are hard to translate in a game simply by nature. Even if a game-like task is doable, it also creates a competitive environment which in community drive systems, but can also lead to cheating or abusive behavior. In short, having a fun to use application is basically mandatory, but using a game to make the users participate can be dangerous and time-wasting in development if the data collection is not practical within a game. Smartparking application specifically are designed to be usable in car, most likely in traffic, while the user is driving. Having to play a mini game to be able to claim a parking spot would be insanely dangerous and most likely result in a ban of the application, or at least would make users run away from it, which is not a desirable result.

At last, the skill learning factor, it can be seen as a more specific approach to the self-benefit factor, where the user picks tasks he does not fully master, in order to increase his skill in such field. The gain is thus completely immaterial and is more personal than community-driven. But such a motivator is also really specific in terms of the platform allowing it, since it must propose a large spectrum of tasks, otherwise the user can only improve a few skills, and if the ceiling on such skill is low, he won't be using the platform a long time. The tasks themselves must also be such that they require some computation from the user, otherwise crowdsourcing wouldn't be necessary or even recommended, since user input always comes with an error margin that can, in some tasks, be higher than what a computer would achieve. In such, the Amazon Mechanical Turk is a really

good example of platform allowing skill learning. But in our case, there is no real skill to be learnt in trading parking spot and the task has to be accessible to all users to allow the application to spread easily. It's then probably unwise to focus on this factor while designing the application.

Having reviewed the biggest factors, we can first exclude the motivators we won't be using for sure : money and skill learning. Those two incentives have clear advantages in keeping the crowd interested, but do not fit well the requirements of our system. Fun is important, but the concept of game for data collection does not apply in this case. Thus the two incentives we will pay the most attention to when designing the application is the community factor and the self-benefit factors. Creating a community willing to share parking spot combines well with the time benefit each user should gain if the system works properly and gathers a large enough community.

2.3 Building a community

We've established that our biggest challenge for our system to be usable is to make many people willing to share their spot, and thus creating a "ParkExchange Community". We can now then start to study the main principles we have to consider in order to develop this community feeling.

Based on the book "Building Successful Online Communities" by R. Kraul and P. Resnick [18], we can identify the biggest parameters to take in account. Their first claim, in accordance with an article by N. Michinov, E. Michinov and M. Toczek-Chapelle [21] is that adding an identity attachment to the community leads to more willingness to participate, and also that clustering populations into groups that share a common property, and even identify the group with a name indicated the membership of individuals to said community increases the commitment of the members of said community.

2.3.1 Identity based commitment

This is really interesting and useful in our case, since we can already create groups with a common property, based on their location. The application could indicate that the user belongs to "ParkExchange Brussels" or any other city in which our system is made available. Of course this must not prevent users from other cities to use the applications in the town, but common sense indicates that the users will use the application most of the time in the city they visit the most. Thus, having named groups, or having the user being indicated that he's using the "Brussels" version of the application should, according to theses informations, increases his willingness to share his spot, since it is for the improvement

of his town on a global scale.

It is also very likely that our application would only be usable in main cities, as opposed to rural environments and small towns, either because there is most likely no need for parking spot exchange in the former, and the user base might probably not be enough to provide enough spots for the later. Thus in a way, our application is already divided into subgroups, for each city it will be available in.

This introduces another parameter of importance for a community to strive for : making the purpose or goal of the community clear and explicit. As we said previously, people looking for parking spots generate up to 30% of the urban traffic [2]. By providing a service that facilitate the parking operation, the goal is also to fluidify the traffic of the city as a whole. Research [5] [18] indicate that having a clear goal that each member of the community works towards is also a major factor in the increase of the sense of belonging to the community, and thus to the willingness to participate and remain part of the group. With this in mind, it should be clear that the goal to improve the urban circulation should be clearly advertised within our application.

One last parameter that could be taken into account is the anonymity. For identity attachment factors, anonymity is also something that improve the attachment to the group. It may seem contradictory but de-emphasizing the individual will strengthen the group as a whole. While this is good news, this factor is however not something we would have worked on since our application would obviously have been anonymous either way, since allowing people to be identified to their vehicle through our application would not have been acceptable.

2.3.2 Bond based commitment

The next set of parameters are called “Bond-based” by the authors. This set contains multiple affirmations that users are more likely to commit to the community when they have or develop a connection or bond with other members. Whether friends they bring into, colleagues that share a common interest or even people they have met through the platform, knowing that those people are members of the community and participate in it , as long as the system provide some way to interact with them.

But in the case of Smartparking solutions based on trade, there is some issues to use those parameters. As said in the previous section, anonymity in our platform is not something we can, or want, to avoid, thus making it harder to provide interaction based on people’s identity. However, one of R. Kraul and P. Resnick [18] claims can still be somewhat useful to use. According to them “Facilitating interactions with “friends of friends”

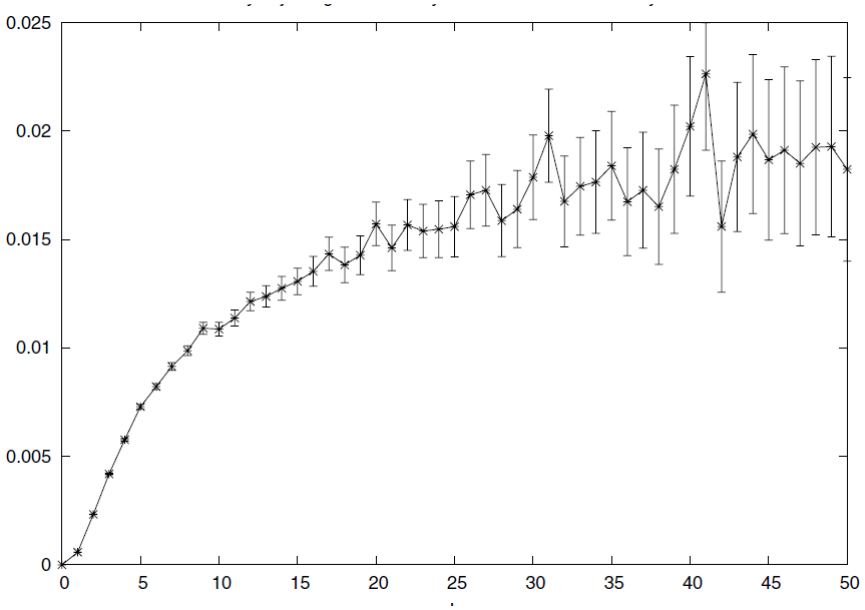


FIGURE 2.2: Probability p of joining a Livejournal community as a function of the number of friend k already in the community [3]

can enhance bonds-based commitment.", with which we can also deduce that simply facilitating interactions with direct friends should also have the same result.

Knowing that, one option to consider for our platform is to allow people to have some sort of friend list, or generating it from the user contact if he allows it, and give priority to friends amongst others when offering a spot. Knowing that you might help someone you know personally seems to be a factor that strengthens the community and the willingness to contribute. Another option we can consider but that would be harder to actually realise would be to indicate, as long as the user wishes it, when a friend of his, or even a friend of a friend, is actively looking for a spot close to him, in order to motivate the user to share his spot. We could even go as far as designing the algorithm calculating the costs (in points) of a parking spot exchange depending on the degree of friendship between the seller and the buyer.

Obviously those are options to consider and not everything can be done in the time frame we have, but it is interesting to keep those options in mind for the future.

2.3.3 Need-based commitment

The following group of claims is probably the main one we have to fulfill to achieve a stable community with our application. Theses claims are called "Need-Based" by the authors of the book, and are obviously related to the benefits people seek while using our system, and the satisfaction level associated. It's not hard to understand that the better our application meets our community's expectations, the more people will be willing

to contribute in return, and the more likely they are willing to keep using it.

Thus the claim "Providing participants with experiences that meet their motivations for participating in the community increases their need-based commitments" is in itself quite obvious, but provides some areas of reflexion. Our application will be using an algorithm to select buyer or buyers whom to offer a parking spot based on multiple criteria, mainly the buyer reputation and credit balance, distance to offered spot and waiting time of each users.

We can have a theoretical approach of this algorithm, and design a version where for example we give priority to the user reputation. With such algorithm, people with a lot of points would be given spots even if they're far away from it and users with fewer credits, typically new users, might have to wait a longer time to obtain a spot. On the opposite side, an algorithm that would favor the distance to the spot could frustrate long time users with good reputation and a lot of points, which in turn might dissuade them from offering their spot again.

As developers, should our community successfully grow and spot exchanges becomes common, one of our first step would be to refine the algorithm by asking feedback after each exchange, or when people refuse spots offered to them, in order to understand if they were satisfied with the exchange, or why they weren't. Was the exchanged spot to far from their location, did they wait too long, etc... This feedback could then help us to improve the algorithm, and hopefully have it creates a better match between parking spots and buyer, which ultimately increases the user satisfaction and thus makes our application respond better to their expectations.

This can also go further. Ideally, each user would be allowed to define his preferences in term of distance, waiting time and reputation. Some people don't mind having to wait more, while others will be happy with a spot several hundred meters away from their selected location. As for the "sellers", they could also choose to only give their spot to users having reach some degree of reputation, or having more points, since those people have proven being reliable, but at the cost of gaining less points while making the exchange.

Since every user need is different, our application will have to evolve to have a better match to the community expectations, through most likely user's preferences and a scalable spot-assignment algorithm, our community would keep increasing [9].

The other claims in this category are less relevant in our study field or can be brought down to simple marketing obvious logic, such as not advertising competitors, increase commitment of our userbase, or rather, prevent users from leaving the community.

2.3.4 Normative Commitments

Finally, the last type of commitment, called "Normative-Commitment", is related to the feeling of loyalty toward a community, and the need to help it to achieve it's purpose and is also linked to the sense of reciprocity, giving back to a community that helped you somehow. This is also related to a claim explained earlier, mentioning that the goal of the application, and it's community, have to be clearly stated. Smartparking application's purpose is to help people to get parked and ultimately reduce traffic congestion in the city. By indicating to the userbase that their contribution helps the city as a whole, and that they can improve their city by contributing, or recruiting others people, it is possible to create a feeling of reciprocity. But in the course to achieve this, some adjustments have to be done, based once again on the claims of R. Kraul and P. Resnick [18].

The first claim indicates that in order to boost these normative commitment, it is necessary to display the community's purpose, as said previously, but we also have to highlight its success to achieve said goal. In a Smartparking application's case, it's not something that can be done easily, since the goal is large scale , traffic congestion won't be solved instantly as soon as the system is released. But after some time, making a study to compare the situation before and after, and hope to see some improvement, that can then be shared to the userbase, to make them realize that they are in some way helping the city as a whole. It would also be interesting to calculate the average parking time of the application's users compared to the parking time of non-users, and hope that the application allows for faster parking and then publish the result to the community, which in turn could help recruit more users and make people more willing to contribute since they can see they are making a difference.

The three other claims emphasize on reciprocity, and provides some interesting ideas to improve the application and the willingness of our users to contribute. When an application displays to the users what they gained by using the application, it should make them contribute more. This could be done simply by indicating how many times the users had parked thanks to the application, and the average time of his search, compared to the average time in the city. A study [8] also recommend to indicate when they gained an over-average benefit thanks to the community, as it can boost this user willingness to contribute to the community.

A more direct approach would be to signal to our users when they can return the favor to someone who offered them a spot previously. For example, if a user X has offered a spot that user Y has benefited from thanks to the application, if user X is searching for a spot and user Y is located nearby, the application could send a notification to this user warning that he can offer his spot (if he is parked, obviously) to the user X, without mentioning his name, simply by saying "A user that offered you a spot recently is looking for

one in your area right now" and having them matched automatically if user Y accepts offering his spot.

2.3.5 Overall

All those combined factors are designed to provide an interesting user experience while using the application, and keep them interested in the application, while giving them more and more reasons to contribute by offering parking spots. Each user having different motivations, it is important to have multiple ways of encouraging our members to be part of the community, and ultimately share spots, since it is the goal of our application. Recruiting people looking for spot should not be hard as it is something many people need on a daily basis. Having those people reciprocate and share their spot as a natural behavior is the real challenge of our application, and all the previous claims detailed before are keys to achieve this.

2.4 User Reputation and punishment

In every community, there will always be abusive members, for personal interest or sometimes simply because they find it fun to annoy other people. It is important for every community to identify these toxic members, usually called "Trolls" in the Internet slang, and to have preventing measures in place to deter them from causing trouble as much as possible. However, it is even more important to be able to distinguish Trolls from normal users causing trouble by mistake, as mistake is typically human. Usually, the biggest factor between the two will be the repetitive nature of the former, thus our system has to be able to detect when a member generates critical negative actions more often than the average user.

In our case, we can identify five critical actions a user can cause harm to another user. Those actions are :

- Offering a false spot
- Canceling an offer while in a trade
- Giving a false feedback as buyer
- Giving a false feedback as seller
- Claiming someone else's car

For each action, we need to create counter-measures to prevent nefarious behavior as much as possible.

Giving false feedback as a buyer or as a seller are problems that can be resolved thanks to a well-though economic system, and thus we will explain how we handle these actions in a following chapter. It is however important to introduce the notions of successful, contested and unsuccessful trades as they will be important for the rest of the section. At the end of a spot exchange, the seller and the buyer are asked for feedback and have to indicate if the trade was successful or not. If both agreed that the trade was successful, the trade itself is flagged as a success, or if both deny it, it is then flagged as unsuccessful. If the seller and the buyer disagree, the trade is flagged as contested, meaning that one of the parties has lied during the feedback.

With these notions in mind, we can introduce the concept of user reputation, which is representative of how trustworthy a user is, and how well he contributes to the community. This score starts at a default value, and increases with each successful trade, with a bonus for selling a spot, and decreases when canceling a trade or being involved in suspicious trades.[1] This reputation would in turn be used in the matching algorithm we described previously to increase chances to receive a spot, as users with a high reputation level would be prioritized over those with a lower one.

Additionally to its impact in the algorithm, the reputation value is also a tool to handle Trolls and dishonest users. One of the first options is to prevent users with a too low reputation level to participate in trade, effectively banning them from the community. This would only happen in extreme case and after several warnings, but can be seen as a last line of defense against recurrent Trolls. However, this does not prevent the malicious user from creating another account. Another possibility would be to create “pools” of user depending of their reputation, and having them interact only with users of the same pool. Thus users with good reputation would only be matched and trade spot with other users with the same reputation level. On the other hand, members with low reputations would be matched with one another until their reputation increases again. This creates a so-called “Prisoner Island” where bad users are confined and prevented to interact with others. This may appear as an interesting solution to protect most of the users from Trolls, but has a big downside.

This solution has been tried and analyzed by Riot Games [4], developer of one of the biggest online game “League of Legends” with a community reaching 100 millions monthly members. Of course our community cannot hope to reach such a number easily, but its points on users toxicity and about the "Prisoner Island" stand still. In their case, players reputation is affected by the behavior of the player during the game, being social, polite and positive increases your score, while insults, threats to other players and

[1] Trades are called suspicious when they might be manipulated by two or more users in order to gain points easily.

misconduct such as leaving a game while it is still ongoing leads to decrease in reputation. According to them, users dropping into the island sees their probability to quit the community increases up to 320% compared to the other users. Indeed being confronted to toxicity nearly all time long is more likely to make you give up on the game and its community. Instead, opting for solutions allowing users to reform gives a better overall growth of the community.

Our case is not exactly similar, but the logic behind still exists. If we put all our bad users together, the frequency at which they will experiment trade failing and having to deal more frequently with dishonest users would lead them more likely to simply stop using the application since it will often not provide them the service they expect.

Instead, the approach used by Riot Games is to allow users to reform notably by keeping them in contact with normal, non abusive members. As for us, it is then more productive to develop solutions to prevent the abusive behavior, by having safeguards against these abuses, which we will talk about later. We will enter the details of our application's development, and in the case of users reputation dropping, not separating them from the other users (except in extreme cases, hence the ban possibility), but showing them the benefits of being a trustful and reputable user.

Chapter 3

Economy

3.1 Overview

As we have now established the foundation of our community and application, and thus have found many ways to motivate our user-base to trade parking spot, it is time to dive into one of the components of those trades, the point system used as economy inside our system.

When we first started analyzing how our application would work, the parking spot exchange was obviously our main component, but for a long time, the economics behind it remained a huge question mark for us, as it was not clear what we were going to implement. However, one question was cleared quickly, as we rapidly found out that using real money for our trades was out of the question, since it could cause a lot of legal issues as our "product", parking spot, obviously does not belong to us, they're most likely public spaces. Thus engaging a financial trade over those spots could be illegal in some countries. At the very least, the management of these spots could have already been given contractually to another company, but even if they are managed by the city itself, monetizing the parking spot transfer could lead to legal conflict [25].

3.2 Virtual Economy

Even without the usage of real money, we can still establish what we will call a "virtual economy", an economy specific to our system, that does have no connection to the real economy and which is only used inside our system (or eventually in conjunction with other CommuniThings applications). Each user has points, receiving a fixed value when they register to the system, and then every time a user offers his spot, he gains points, and in return, buying a spot cost points, meaning you can't only buy spot without have ever offering one. The basic idea is quite simple, but to make it work we have to consider a lot of criteria.

SmartParking is still a relatively small world and few applications have tried to achieve complete trade of parking spot, with a seller and a buyer. A lot of applications, like ParkTag [28], tends to only detect when users leave their parking spot and notify all the other users that a spots is available, and then these users can reserve it, but without any insurance that someone else (within the community or not) will not precede them, making the spot unavailable. Since there is no real trade between the users, no economic system is required, and when someone fails to obtain the spot they wanted, the user have to deal with it, the reservation is not guaranteed and no compensation is given to the user This is only fair since he didn't invest anything into the spot, except time and gas, but those are not related to the spot exchange in itself.

Our application should offer a similar service, using CommuniThings data to determine the level of occupancy of a street or other parking zone within a city and then redirect users towards the matching areas with available free parking spots. This functionality does not require a functional economy to work, but does not cover entirely what we want to achieve. Fully committed parking spot trades, with buyers and sellers matching with each other's, having the seller waiting for the buyer to arrive and give him the spot, requires a lot more coordination and has to be rewarding to the seller, otherwise they would simply leave the spot, but then also has to be way more reliable for the buyer, as on the other hand they are investing in this spot, no only time and gas, but also virtual currency. As such, we have to do everything in our power to ensure that a trade is successful so as to ensure the buyer that he will get his spot, otherwise he could see the application as a form of scam, which would be obviously rather bad for advertisement and recruiting.

Other applications, or theoretical works have already worked on parking spot sharing and trading, the most notable example being the late KurbKarma [19] application. We will identify their trade process and the economy behind, as well as the system named CrowdPark [38], also designed as a social parking application, but it is unclear if the theoretical work done by its author was evaluated in real.

3.3 Existing virtual economies in SmartParking

We are not pioneers in the creation of of an application designed to facilitate parking through spot trading. KurbKarma and CrowdPark are two applications that share a similar purpose to ours. Both used virtual economies that could be interesting starting points for the development of our own economic system.

3.3.1 KurbKarma

Kurbkarma used “KarmaKredit” [19] as currency. Its economic system was really straightforward : buying a spot costed 2 karmakredits, while selling the spot cost one. One of the first implications is the symmetry of the system. The purchasing price is higher than the selling gain. If the trades were symmetrical in cost, the amount of points into the system would have remained the same after each trade, meaning the only reason for account balance to disappear is when users stopped using the application. But it is also necessary to consider that it is logical to offer credit to new users, thus creating a constant currency input in the system. For KurbKarma, this value was set to 10, meaning that each new account was given 10 karmakredits, enough to buy 5 spots.

If unregulated, the total value generated by the account creation mechanism could have led to the hyperinflation of the economic system, but applying regulations goes ultimately against one of the major objective of the application, that is to recruit more and more people in the community to expand the amount of available parking spots.

This problem can however be reduced, if not avoided, by using asymmetrical trade prices, which Kurbkarma does. In short, the buying prices being higher than the selling gain means that the seller won’t receive as many credits as the buyer is spending. This is not uncommon with virtual economies, for example the MMORPG "World of Warcraft" has a 5% difference between buying prices and seller’s gains while using the in-game auction-house, meaning that buying an item 20 gold (the name of the WoW in-game currency) will only result in a 19 gold gain for the seller.

Of course, we cannot really use "World of Warcraft" as an economic model, since it is much easier to obtain currencies through a game, as players are rewarded for doing any action inside the game and they have much more options to spend their currencies. In our system, users won’t be generating any credit except when trading a spot, and the only possible expense is buying a parking spot.

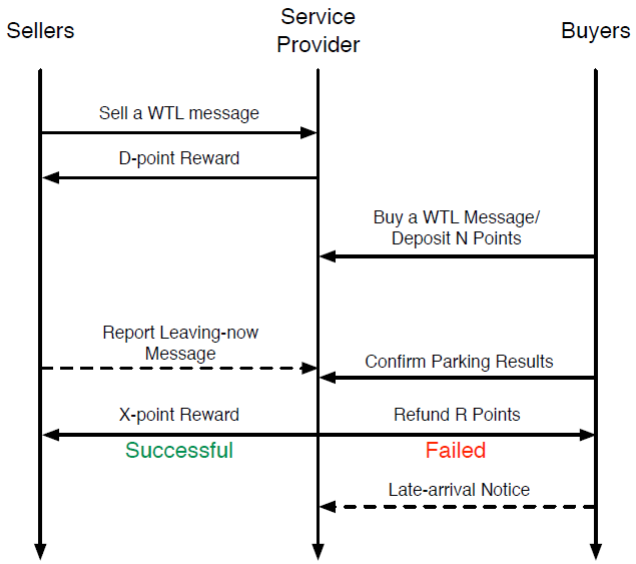
Kurbkarma asymmetrical costs are rather extreme, with a 100% difference between the buying cost and the selling gain. This is somewhat compensated with a large generosity with the free credits gained on registration, but this bonus quickly fades away. Having to sell two spots before buying one might prove frustrating, or requires a deeply engaged and motivated community. To protect themselves against scams and sellers leaving spots early, ruining the trades, Kurbkarma used a reputation system, allowing users to rate the members of the community they traded with, and allowing the users to check the seller’s reputation before buying his spot, thus making it engaging for people to complete the trade in order to improve their reputation and sell spots easier and faster in the future.

3.3.2 CrowdPark

The other application that attempted to implement spot trading is CrowdPark [38]. Designed by computer science researchers at the University of Massachusetts, its concepts are very close to our own, having an application to allow users to trade parking spots with ultimate goal to reduce traffic congestion in the city center. However, one major difference between their work and our is the time-line of the trade. Our objective is more to respond to an immediate demand of users looking for a spot as they're driving close to their destination, while CrowPark opted for more planning, having their users indicates several hours in advance when their current spot would be available, typically when they leave work, and having the buyers reserve the spot before even leaving their home, and matching their arriving time to the depart time of the seller.

CrowdPark's economy is also asymmetrical. Sellers receives a reward D at the moment they sell their spot, even if no match is found yet, and the reward will be kept even if nobody buys the spot. D is fixed, the seller knowing in advance how many points he will gain. There is however a bonus X if a buyer is found and the trade accomplished successfully. This bonus is lower than D, the authors mentioning it being approximatively a quarter of the fixed reward, but it could be varying depending on the location of the sold spot, downtown spots being more valuable than suburb spots.

On the other side of the trade, the buyer can reserve a spot, and pays N for it. The transaction is executed instantly when the trade is registered. Other big difference between CrowdPark and our own system is that the confirmation of the trade relies only on the buyer, while we ask confirmation and feedback to both parties of the trade (more on this in Part 2). If the trade is successful, the sellers gets their bonus X, otherwise, the buyer can obtain a refund R, not covering the entire cost (N) of the trade. The process of spot reservation and currencies exchange is detailed in the following figure.



3.4 Dynamic Pricing

The two examples given above shares both a well known fixed price and gain for spots and all spots are of equal importance (with the eventual small possibility to have a variable bonus in the CrowdPark application). In reality, downtown spots, or parking spots close to points of interests like shopping areas and cultural centers are more valuable than spots in a suburban area. In a similar way, spots values vary during the time of the day, more users are looking for a spot in the afternoon than at 3 A.M. , by "simple" law of Supply and Demand. Such variations could lead to a mechanism we will call "Dynamic Pricing", where the cost of a spot and the gain to the seller are variable, and better reward the users that offer a spot in highly demanded areas, during peak hours, etc. We found two systems using dynamic pricing that looks interesting to us. First the pricing system in place for parking in San Francisco, SFpark, and secondly the UberX system designed to react to variations in supply and demand. Even though Uber is not a smartparking system, it possess enough similitude in its pricing strategy to be relevant in our case.

3.4.1 SFpark

SFpark[1] is the name of San Francisco parking pricing program, initiated in April 2011. Through the usage of sensors detecting the occupancy of parking spots in San Francisco streets, the price of parking vary, depending on the average free/occupied parking spots ratio of a street, each one independently of the others. Before SFpark, the price of parking spot in SF downtown was fixed at 3$ an hour, and some of the biggest and most busy street were nearly constantly at full parking capacity, while some nearby streets might

[1] http://sfpark.org/

have 40 or 50 percent of free spots.

The purpose of SFpark is to create a better distribution of the parked vehicles, through variating parking prices depending on the occupancy of the street. Ultimately, the objective of the system is to have on average between 60% and 80% of the parking spot of a street occupied [29]. If a street is above this value, the price of parking per hour in this particular street increase. Oppositely, if the value is below 60%, the price decreases.

With this variable price system, the purpose is to lead people toward cheaper places, even if this increase their walking distance. If all the streets have an average occupancy, it means that every user should be able to find a spot matching his demand more easily, as no street has a 100% occupancy.

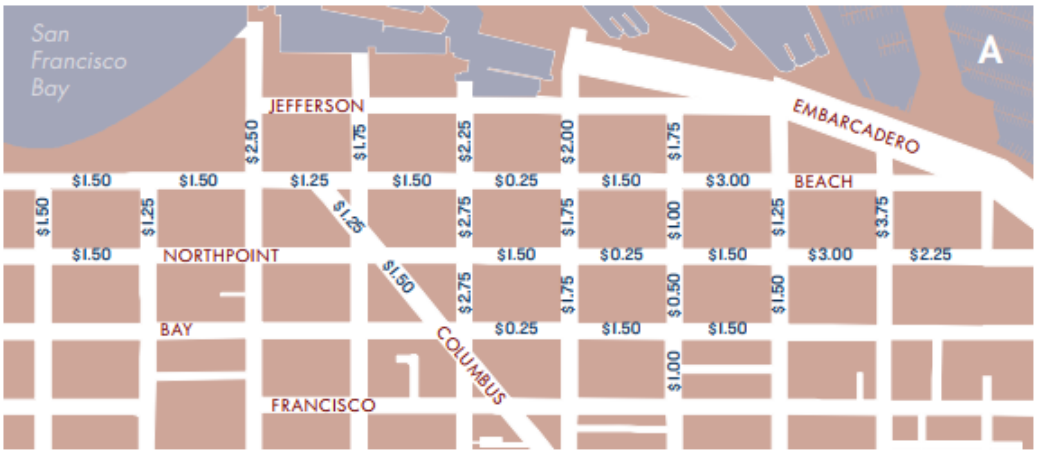


FIGURE 3.1: San Francisco streets dynamic parking prices, May 2012 [29]

There is however some issues with the SFpark system. First of all the price of the installation, estimated at 18 millions USD, for the 7000 parking that were fitted with sensors [24]. San Francisco being a huge city it was possible to afford such investment, but this is not the case for many cities around the world.

Additionally, the system is not flawless as two parallel streets in the targeted occupancy were found to have a very large price difference on average with spots in one street costing more than double those in the other street. Several other factors, such as the abuse of reserved spots for disabled people, also have influence on the price and the occupancy of streets, and as of 2013 the SFpark system was not yet taking those sufficiently into account. But the idea of having the prices depending on the geographic location of the parking spot remains sound and coherent, and is something we have to keep in mind should we ever implement a dynamic pricing solution for our SmartParking application.

3.4.2 UberX

The system used by Uber[2], named "Surge" pricing [6], has for objective to increase the response when the demand of service increases heavily due to some events or peak hour of transportations. Even if the service provided is not the same as us, transportation instead of parking, the context is similar as our peak times and hours are similar to those of Uber, the difference being our public target. Uber target people not willing, or not able, to use their own vehicle to reach a destination, while we target the opposite, the one who drives to the destinations and who will have to park their vehicle in areas where parking is not trivial.

Uber's "Surge pricing" is a single value that impacts all prices of the service. This value is calculated in real-time, depending on the demand, increasing by 10% increments. The default value, 1.0, means that the prices are in their normal state and demand and supply are balanced. When demand starts to outmatch the supply, for example at the end of a concert or football match, this surge value starts increasing. As long as the demand gets higher, the surge value keeps increasing. In their article, J. Hall, C. Kendrick and C. Nosko [13] observed the variation of the supply and demand of the UberX service during a popular event. The following figure shows the number of users opening the Uber application in Madison Square Garden area, during an Ariana Grande concert.

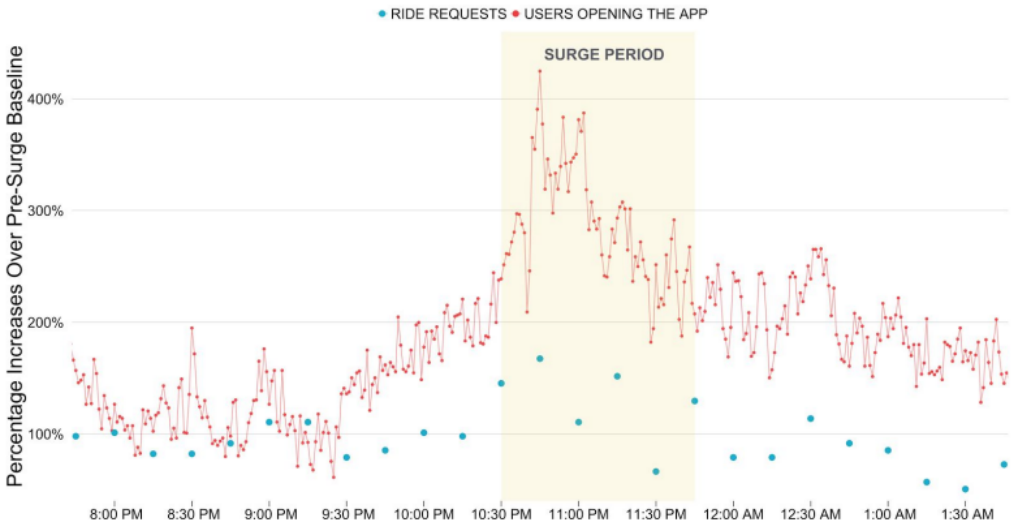


FIGURE 3.2: Demand for Uber Spikes Following SoldOut Concert on March 21, 2015 [13]

The surge period is highlighted and we can clearly identify the moment, at the end of the concert, when the demand starts to rise. This leads to Surge value increases, up to 1.8 for five minutes, meaning a 80% prices augmentation. During the 75 minute long surge period, the Surge multiplicator was a 1.0 for 40 minutes, increased by one tens every 5

[2]https://www.uber.com

minutes. The increase in prices was followed by a rapid increase in service suppliers (Uber drivers), which is also clearly highlighted in the following graph, representing the amount of Uber drivers responding to this high demand in the area of Madison Square Garden.

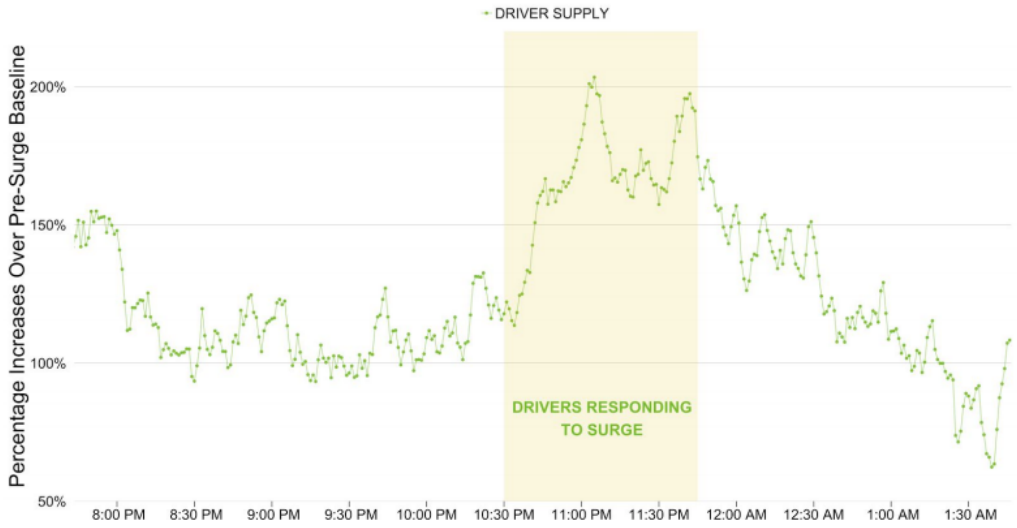


FIGURE 3.3: Uber DriverPartner Supply Increases to Match Spike in Demand [13]

As the authors pointed out, those values by themselves cannot be claimed as proof that the price increase led to the the supply increase, indeed Uber drivers could have been moving to this area by themselves knowing that the concert was about to end and more of them were ready to respond during the surge period, however the correlation between the two graphs is too strong to be ignored.

Another situation studied by J. Hall, C. Kendrick and C. Nosko [13] reinforced their hypothesis that the surge pricing effectively increases the supply during peak hours, while also pointed out an issue with the system. During New Year's Eve 2015, Uber surge pricing was in effect, even reaching a 2.7 value (170% prices increases) until 1 A.M., when the surge system broke down, leading to a surge value dropping to its default 1.0 . New Year's Eve is also a remarkable period of low supply, as many people are busy with their own celebrations and are not willing to provide the Uber service, except when the prices are getting really high, thanks to the surge value. But with this value dropped to 1.0 due to a glitch, very few Driver-Partners responded to the demand, and many people where left with their request unfulfilled. This is highlighted in the following set of graph, where we can see the amount of requests increasing, but this time, without the surge pricing the supply remained low, and the request completion time heavily increased. The percentage of request completion dropped below 25% during the period, while it stayed a 100% during the surge period of the Madison Square Garden example.

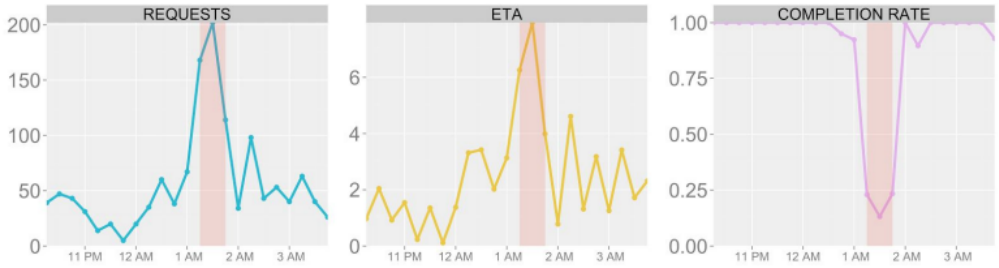


FIGURE 3.4: Vital Signs of a Surge Pricing Disruption on New Year's Eve (January 1, 2015) [13]

This example leads to two reflexions. First, it is evident that surge pricing as a mechanism to increase supply works, and that we could use a similar system to encourage users to share their spots during peak hours and in heavily demanded areas. But on the other hand, it could lead to users delaying their offer, or not offering at all, when the surge is not in effect, either due to a problem on our side (glitch, system failure) or simply because the demand is not high enough.

In any case, it is important to evaluate the pros and cons of each pricing system, dynamic or static. Obviously static pricing is easier to implement and is straightforward for the users, while dynamic pricing is a solution where architecture and algorithms, while preferable on average, may lead to discrepancies like Uber solution, in terms of peak hours, mobility and traffic congestion.

Chapter 4

Security & Privacy

4.1 Overview

Having discussed and presented options related to the Community development and the economy behind the parking spot trades, it is important now to take some time to introduce security and privacy concepts and measures we have to acknowledge and consider in the development of our park sharing application. Indeed, contrary to many of the SmartParking systems we presented before, our application will need to process and store user's private data, such has names, license plates and timestamped locations. Storing of these informations means that we have to ensure that these data won't be distributed freely without user consent, and that our system will have appropriate security level to ensure that attackers won't be able to access them.

Before we move to the implementation of the application itself, we have to look at good security and privacy practices in the mobile development software process, with the aim to provide some guidelines for the implementation of secure mobile applications. The field of mobile security as a whole however, is considered outside the scope of this research, so we will not cover issues related to malware or operating system vulnerabilities.

4.2 Identifying security risks and mitigations

The OWASP Mobile security project [27] is a collaborative research project that aims to provide up-to-date information about security in the mobile space. In 2016, the project published an updated list of the ten most significant mobile security risks, based on the original list from 2014. The community-driven side of the project offers interesting insights on which issues mobile developers are struggling the most.

1. **Improper Platform Usage**
   Mobile Operating Systems offer numerous capabilities that can be leveraged in application code through dedicated interfaces. Some features come with strict guidelines and security risks might arise if the application does not respect them. For

example, an application might rely on the platform providing an interface to securely store local data, such as KeyChain on iOS. If this application fails to leverage such feature or mistakenly copies data from the secure storage to local public storage, sensitive data may be leaked.
Mitigation: Make sure platform specific features are properly understood. Adhere strictly to the documentation provided by the manufacturer.

2. **Insecure Data Storage**
   The use of encryption for local storage of sensitive data is as mandatory for local storage as for network communications. Local storage on mobile devices cannot be considered trusted since the mobile aspect makes it more likely that the device could be lost or stolen. On top of that, numerous applications typically cohabit on the same device and some third-party application or malware could easily gain access to the file system. Furthermore, the platform itself may leak data during normal operations without the developer's knowledge through internal processes such as caching, buffering or logging.
   Mitigation: Understand the implications of URL or Copy/Paste caching, application life cycle, cookies management and logging on a platform specific basis. Use encryption on local files and store as little information as possible.

3. **Insecure Communication**
   This category encompass all vulnerabilities that makes possible the interception of sensitive data while it transits over any kind of network. Mobile applications rely almost always on a client-server communication model over the Internet that needs to be resilient to data leakage or any kind of tempering. Specifically, risks in this category may concern *data integrity* where data could be altered in transit, *data confidentiality* where sensitive data could be obtained by observing traffic and *origin integrity* where an attacker could impersonate a legitimate user.
   Mitigation: Make the use of SSL/TLS encryption for all Internet communications mandatory by default, with appropriate cyphers and key length. Use trusted certificates, never allow self-signed certificates and warn the user if there is an invalid certificate issue. If appropriate, use a second layer of encryption on top of SSL/TLS before transmitting sensitive data.

4. **Insecure Authentication**
   An application might be vulnerable through a lack of trusted authentication, for example, if it is allowed to execute a restricted API call without providing an access token. If the server fails to properly authenticate the sender of a request it might result in anonymous execution of restricted features. The typical consequences being exposure of private data. Denial of service attacks are also facilitated by such vulnerabilities.

Mitigation: Ensure that authentication checks are performed for all possible requests on the server, and only load private data on the device after successful authentication. Perform authentication checks more than once when processing a request in multiple stage to avoid a single point of failure. Avoid caching of passwords or any other data that could be used to bypass authentication schemes. Require strong passwords. Do not rely on spoofable informations for authentication.

5. **Insufficient Cryptography**
   Mobile applications may rely on outdated or weak encryption schemes that can be easily broken by modern techniques. However, encryption can also be compromised by external factors even if a strong encryption scheme is used, if keys are mishandled or leaked for example. Some applications try to deploy custom encryption tools but it is generally accepted as a bad practice by the community since it is a form of *security through obscurity*. The OWASP authors also warn to not rely on built-in encryption tools for the application code as they can be bypassed trivially with jail-broken devices.
   Mitigation: Use strong and up-to-date encryption standards. Try to plan for future improvements in encryption breaking techniques that may render some encryption algorithms obsolete sooner than later.

6. **Insecure Authorization**
   An application exposes authorization vulnerabilities if it fails to check in a trusted way that a particular user has the proper permissions to perform a particular action. This category is linked to "Insecure Authentication" as authorization obviously requires authentication. Risks in this category could also stem from not validating input data from the client, which could allow an attacker to exploit legitimate requests parameters in order to expose private informations.
   Mitigation: Crucially, the authorization process must rely exclusively on trusted information from the backend rather than on data coming from the client themselves, which would be trivial to manipulate for a potential attacker. It could be advisable to perform additional checks on all requests to ensure they could reasonably have come from the same identity.

7. **Client Code Quality**
   This category concern all risks that directly result from an implementation error in the client code such as buffer overflows, memory leaks or insufficient validation of user input on the client side. Consequences may be severe as attackers can leverage such vulnerabilities in order to compromise the entire mobile device, gain access to large amounts of personal data not limited to that specific application. Buffer overflows are typically used to obtain remote code execution on a targeted device.
   Mitigation: Maintain consistent and healthy software development practices. Particular attention must be given in setting the size of buffers in languages that do not

provide automatic memory management. Use static code analysis tools to detect common buffer overflow and memory leak issues.

8. **Code Tampering**
   By design, mobile application clients run in an environment that is in no way under the developer's control. Some attacks can be performed by modifying the application code itself, either directly inside the software package (bytecode) or through the runtime environment. This category cover risks related to memory manipulation, executable modification or configuration misuse.
   Mitigation: The application should be able to assess its own integrity at runtime based on some information known at compile time. The application could also perform checks in order to detect if the device has been jailbroken or rooted and take appropriate actions if so.

9. **Reverse Engineering**
   As stated above, mobile applications are susceptible to code modification. Thorough analysis of the client's code can realistically inform attackers about data structures, algorithms and other assets contained in it. For example, String constants and comments can be easily retrieved if no steps were taken to prevent it, which in turn might help an attacker gain valuable insight in the inner mechanism of the application.
   Mitigation: Use code obfuscation tools to make the code less understandable from an external point of view. Such tools will always have an impact on performance so it is important to find a good balance.

10. **Extraneous Functionality**
    Developers often includes hidden features in the code, in order to speed-up testing or deployment. Such features are typically not intended to be released in production but some artifacts are often left-over. Furthermore, mistakes do happen, the canonical example being a commented password inside the code or worse, the entire authentication process was commented out during testing and was not reactivated before release.
    Mitigation: Inspect the code thoroughly before release, possibly with the help of unit testing and possibly with symbolic execution tools. Make sure there are no deprecated or hidden endpoints on the server side that were left-over from development.

4.3 Testing

An healthy architecture with appropriate security measures, while required for secure mobile applications, is no guarantee for security. Some tools like static code analysis and unit testing can contribute greatly to the detection of vulnerabilities, even early on in the

development process, but entire categories of software bugs only manifest themselves depending on context: network congestion, system load etc. This is especially true for mobile applications due to the mobile nature of the targeted device where the environment plays a greater role than in traditional applications: device location, battery charge, network connectivity, sensor inputs are all factors that can influence mobile applications behavior.

Another crucial aspect of mobile application testing is the large number of different platforms, operating systems, software libraries and hardware components on the market. Interoperability between these components relies on an increasing number of abstraction layers that could lead to software failures rather than the mobile application itself. This problem is exacerbated by the short update cycle adopted in mobile ecosystems. Therefore, testing is required to ensure mobile applications behave as expected when running on different devices. Unfortunately this entails a high cost for software developers.

A study by H. Muccini, A. Di Francesco and P. Esposito [22] from the University of L'Aquila, Italy, concluded that mobile applications are indeed sufficiently different from traditional ones to warrant specialized testing techniques, due to their *mobile* and *context-aware* aspects. The paper identified many challenges related to mobile apps in regards to performance, reliability, energy usage and security. Specifically, the paper highlighted that a typical mobile device may connect to different networks with different security levels, such as public/private wifi networks or cellular networks. The paper argues that dedicated testing is required to ensure applications do not transmit sensitive data in an insecure manner over potentially compromised networks.

The comprehensive testing of mobile applications is therefore challenging. Because the variability in context and components is so high, it is unwise to expect that manual testing will ever be sufficient to guarantee a safe level of dependability. Automated testing tools seem to be the most promising answer to many of these challenges, particularly the notion of *Testing-as-a-Service*. [23] and [11] present models to perform mobile application testing in the cloud which has numerous advantages over other models, namely:

- **Cost-effectiveness**: Cloud computing, in its essence, is about sharing computing resources which brings down costs. Cloud based models can easily include a "Pay as you Test" billing scheme as presented in [11].
- **Scalability**: Cloud based testing relies heavily on virtualization techniques, which makes the daunting task of covering numerous devices much easier.
- **Availability**: The cloud model offers high availability by design, without the need for dedicated hardware.

However, the *Testing-as-a-Service* paradigm also entails some new concerns, notably in terms of privacy. If the tests performed contain actual user data, they should be thoroughly anonymized to prevent any personally identifiable information to reach the test environment in order to avoid any issues if the service is compromised in any way.

We expect that *Testing-as-a-Service* will soon become standard practice in the industry, which would undeniably raise the level of security in the entire mobile software sphere, granted that dominant players like Google Play Store or Apple's AppStore enforce automatic testing on their catalog.

4.4 User expectations and perceptions of privacy

Smartphones are hosting a large amount of personal information that typical users would generally consider as private such as emails, text and voice messages, contact informations and photos. In many ways, the smartphone has replaced the personal computer in people's lives, but the mobile nature of the devices and the vastly different software ecosystem and distribution model make direct comparison of privacy implications challenging. Users might focus on specific expectations in regard to the privacy of their information that they inherited from their use of desktop computers that, in reality, are not met in the mobile computing space.

Studies have shown that average smartphone users indeed demonstrate significant privacy expectations in regard to their smartphone usage. For example, A study by J. King from the University of Berkeley, California [17] draws a parallel between a person and an application accessing a mobile phone and the notion of trust level. Study participants mostly reported that they would have no issues with a person they trust accessing their devices, but this person would still need to give a valid reason to access personal informations. In principle, the same applies for mobile applications: users are generally ready to share information if it is contextually relevant. However, participants expressed discomfort when confronted with application behavior that did not met this expectation. Applications requiring permissions to access sensitive but unrelated data are perceived as invasive or even deceitful by users. But data collection and sharing to third parties is often at the core of application business models.

The study also shows that users have troubles evaluating what kind of information applications actually have access to. At the time the study was done in 2012, both Android and iOS applications could access information that many users consider private such as photos and texts under the default settings and many participants were unaware of that. Although permission settings are more strict nowadays, usability remains an issue for users with little technical knowledge.

Participants also reported feeling powerless when faced with mobile applications *Terms of Service* or *End User License Agreement* that are both time-consuming to read and hard to understand. These TOS represent a kind of "all or nothing" contract between the user and the service provider in which the user has no power of negotiation, the only alternative being to not use the app. As a result, very few users actually read TOS or EULAs, leaving them exposed to unexpected or hidden data collection. For example, when researcher guided participants during the installation of "Fruit Ninja" and clearly exposed all the permissions the application was requesting, participants felt uneasy and deceived.

Many participants reported that they place their trust on reviews, reputation and the distribution platform rather than on applications themselves, in the belief that such platforms would not allow dishonest applications. Unfortunately the review process of such platform is rather opaque and provides no such guarantees to the end user, at least not in a transparent way.

However, both [17] and [31] highlight that many users still use applications that they perceive as valuable even if their expectations of privacy are not fully met. The global feeling expressed by users is a lack of solutions to the problem. Users are generally not inclined to search for technical solutions such as tracker blockers and simply favors the convenience of mobile applications. It is unclear what long term consequences this distrust from mobile device could have in consumers, although researchers suggest that their could be negative psychological and health impacts through the stress resulting from the cognitive dissonance associated with the invasion of personal space.

The main conclusion is that users are increasingly aware of the dynamics of the information economy and their role in it. People generally agree to share their data if they agree to the purpose and have given their explicit consent. However, complicated EULAs and TOS hardly count as informed consent, so developers should put more emphasis on what data users are really sharing during usage. The "creepiness" induced by the discovery that some applications are more than they appear (like "Fruit Ninja) can be mitigated by making sure the user has some form of control over shared data.

More generally, we argue that application business models should not rely exclusively on data sharing. It is much more beneficial for all parties involved to first and foremost bring value to the user through the collected data, as it is then contextually justified to do so. If more value can be extracted from those data by sharing them with other actors, express individual consent is mandatory to maintain a trusted relationship with users.

Chapter 5

Reliability

5.1 Overview

When talking about Crowdsourcing, one of the challenges that rapidly comes to mind is the reliability of the crowdsourced data. How trustful we can be toward our users and mostly, how we can prevent users from submitting false data ? This is a challenge that every crowdsourced platform faces to some extent, and with our smartparking application, we are not exception.

However, due to the nature of the system we aim to develop, we are not as impacted by false reports that some more traditional crowdsourced platforms, as the data we requires from our user are very specific, at least for the first iteration of our prototype. But reporting should still have its place in the future of "ParkExchange", and it is then important to already consider the options at hand to improve the reliability of our crowdsourced information.

5.2 Seller's reliability

For the purpose of our application, the main crowdsourced information is the availability of parking spots, when users decides to sell their spots. The obvious case of false reporting is user reporting an available spot when they are not actually parked. Of course, since there is trading involved, and currencies are being exchanged when selling a spot. We have means to dissuade users to make false sales through economic loss (even if only virtual ones), but that does not mean we cannot try to figure out automatic ways to determine a false seller, before he actively causes harm to another user by selling him a fake spot.

We already introduced CrowdPark [38] in a previous chapter. As a spot sharing application, or prototype, they faced similar issues, notably with the seller's reliability. They

develop two solutions to handle this issues that could prove useful, that they named respectively “SpotCheck” and “ActCheck” .

SpotCheck

The idea behind SpotCheck is to force the potential seller to take a picture with his phone of his own license plate, add a geo-tag to the picture and then send it to the server, when willing to sell a spot. This confirms that the seller is at the spot he claims to be with his parked car. However, this solution presents several issues.

First of all, this is time-consuming for the user, requiring him to step out of his car and to perform an additional action to simply sell his spot, which is counterproductive. When we identified the motivation being part of a crowdsourced community in the “Community” chapter, designing a fun system was a big part of a crowdsourced application’s success. Even though sharing a spot is not really fun strictly speaking, and is not too constricting to the user, as long as initiating a trade remains easy. Adding several steps to the process in counter productive, and could dissuade users to share their spot. Indeed, as a small example, who would really be willing to step out of his car and take a picture of the license plate when it is pouring outside.

The other issue with such a system is the picture itself, and the license plate it contains. Automatic Recognition of textual information in picture data is a long standing challenge in computing. Nowadays, ANPR (Automatic number plate recognition) systems are getting better and better, but leaving it fully automatic without any sort of supervision remains unsafe. According to the CrowdPark developer, in 2012 the ANPR reliability was around 10% [38], way too low to be of any use. Even if we consider that the ANPR technology has improved over the last five years, it remains too much of an uncertainty to be used. The alternative solution proposed by CrowdPark was then to use human-based resources, typically the Amazon Mechanical Turk, we mentioned previously. With the AMT, they achieved a nearly perfect hit rate with their plate’s recognition, but at the cost of lengthening the process, which is not acceptable for our concept. With their system, users sell places several hours before actually leaving the spot, thus delaying the trade for 5 minutes is not too problematic. Our application being much more realtime than CrowdPark, asking our sellers to wait up to five more minutes before even confirming that they could sell their spot would most likely be a killing blow to our application.

ActCheck

The other option proposed by the CrowdPark team is sensor based. In a way it is similar to the first practical task we worked on in the course of this thesis, the small application

collecting sensors such as accelerometer in order to determine later on if the user was walking, driving or standing. The idea CrowdPark developed is that the user should most likely be walking before selling his spot, and driving after the trade is over.

ActCheck requires the application to keep being active as a background process once a seller initiate a sale. It then constantly computes the sensors data and uses an algorithm, in the case of CrowdPark, the JigSaw algorithm [20], and determines the user's activity. Jigsaw is based on the accelerometer and determines the user's situation through the vibrations detected by the device. Using mostly the accelerometer provides also the advantage of requiring less battery usage than other sensors or even the smartphone's camera.

If the algorithm then detects that the seller is performing an action he should not be doing , mostly driving before having completed his trade, he can be flagged as a dishonest seller, and his spot could be removed from the pool of available spots, if it had not been purchased yet, or in the opposite case, warn the buyer in time that the spot he bought was in fact non existing. The buyer then gets his credits back and can purchase another spot.

ActCheck present two issues, first, when the algorithm assume that the user is driving when he is not (false positive), for example if he instead takes a subway on his way to his parking spot, and the second with malicious sellers being aware of the ActCheck prevention system, trying to cheat the system by simulating expected behavior.

For the malicious users, a study [34] indicates that imitating a walking pattern when actually using some sort of vehicle is extremely hard to perform. By using mostly the accelerometer, with the addition of GPS information to refine the data, researchers have been able to determine with a 99.9% precision when a user is not walking, even if he pretends to. The remaining issue is then the risk of false positive. In this case, there is a risk that someone using public transportation could then be identified as a dishonest seller when he is not. The CrowdPark team suggested but without having actually tried to implement such system, that by coordinating sensor's values, GPS information and public transportation routes and schedules, it should be possible to even detect such situations and thus determine with great certitude when a user is actually driving when he should not be, and flag him as dishonest.

Since CommuniThings is working on a similar algorithm with very good results concerning the activity detection accuracy, we could use it in our development, even though it is unclear at this stage how to reliably differentiate a car driver and a bus user for example, as both case generates similar contextual patterns. SpotCheck would be inappropriate for our design, as it delays the licence plate recognition process significantly

while sellers are expected to have a short timescale when they sell the spot. However, ActCheck concept is a good option for our design, being very close to real-time. With some improvements, and in conjunction with CommuniThings algorithm, a similar system to ActCheck could then be used in our application as a measure against dishonest sellers by checking that a seller is not currently driving for example.

Summary

Social and economical aspects, alongside security, privacy and reliability are the main aspects to consider when relying on crowdsourced information. Together, they represent our answer to our research question "What are the major themes to consider when relying on crowdsourcing for a Smartparking system ?"

Existing smartparking solutions rarely leverage the benefits of a strong user community, these instead rely on "smart" infrastructure to provide a reliable service. Based on the theoretical background we exposed in this first part, we will now describe in detail the prototype of a crowdsourced smartparking solution that obviates the need for new infrastructure by having interaction between users at the core of the system.

Part II

Software Development

Prototype Application

Chapter 6

Context & Analysis

6.1 Context of the application

Having reviewed SmartParking concepts with and without a crowdsourced method of data collection, it is now time to take a practical approach to the parking problem by developing our own application, that will include the crowdsourced elements we explored before. This part of our master thesis is thus dedicated to describe our though and work process in the development of the application, based on elements we described before in our field review.

6.1.1 Parking spot exchange model

The main objective of this project is to propose a model where users can sell and buy parking spots to each others directly, with a high degree of confidence that the transaction will be successful for both parties. This entails that users can be reliably located and identified, that users have a way to plan where and how they will find parking space, and that users cannot benefit at the expense of others. The economic aspect is not negligible to ensure an equilibrium between offer and demand as from a practical standpoint there can be only immaterial benefit in signaling a free spot. The model should also account for the fact that dealing with the real world evolves uncertainties and unreliable data or users should not result in critical failure.

6.1.2 Buyer/Seller Match algorithm

As opposed to some other proposed park sharing models like CrowdPark [38], we do not intend to match offer and demand in a free market style but rather through the use of algorithms to optimize the matching between sellers and buyers. We argue that it is indeed better to restrict the user's freedom of choice in order to lift the weight of decision from the user as well as making it more difficult to exploit the system via collusion.
The matching algorithm needs to take into account important and simple parameters like distance and wait time but could be enriched by using traffic data in order to predict more accurately the estimated time of arrival of a buyer for example. More specific information like parking spot size or some form of user reputation could be used as well. Corner cases and abnormal situations also need to be defined.

6.1.3 User Experience

Any crowdsourcing application relies on its user base at its core to work properly and it is especially true in the context of park sharing. There is indeed a critical threshold of adoption below which this application will simply be useless because of the lack of spot offers. We should therefore strive to design the best possible user experience in order to attract new users rapidly and crucially retain active users while the community is still small and the benefits of using the application disputable. This is accomplished in part through a particular focus on user interface and transparency by keeping the user aware of possible actions and their effects.

6.1.4 Deliverables

In order to define more clearly the scope of this project, we agreed with CommuniThings to work on three parts.

Parking Sensor Module

This module extracts data from mobile phones sensors such as accelerometer, magnetometer etc. Basically a logger, the main purpose is to use machine learning techniques on the extracted data in order to determine the feasibility of the automatic parking detection feature mentioned previously.

This module should be simple in its design while being compatible with many brands and models of phone. This is crucial because different phones have different sensors which might affect the precision of collected data. These data should reflect the variability in sensors to guarantee the algorithms trained on are general enough to be useful.
Usage should be manual, where the user explicitly records periods of activity in different contexts such as walking, sitting, driving and of course parking.
Its intended user base is exclusively limited to members of CommuniThings. In the future, its core features could be reused along trained algorithms to implement automatic parking detection.

StopBuy Mobile Application Module

The StopBuy mobile application is currently in use by the general public in the context of the StopBuy platform deployment in Mons. It supports important features such as a dynamic map with tokens, GPS guidance and connectivity with CommuniThings services. The application is in active development.
We created a branch in the application source code to implement new features related to parksharing such as being able to request/offer a parking spot from/to the community as well as a complete end-to-end parking spot exchange. We also implemented dynamic map notifications that displays a token when a newly freed parking spot is detected.

StopBuy Web Application Module

The StopBuy platform backend services are split between different processes, the main one being a web application answering the requests from the mobile application. We also created a branch in the web application source code to implement the necessary features to make the mobile application module work as intended, such as new API routes and associated controllers, database entities etc.

6.2 Analysis of the required system

On top of usual software engineering concerns, the development of a "Smart City" application requires careful analysis of real world considerations. In this section we present in detail the parking spot exchange model we designed in collaboration with CommuniThings. We discuss practical, economical and technical aspects of the solution. Finally, we expose our point of view regarding activity detection.

6.2.1 Parking Spot Exchange Model

Seller's perspective

Once a user is registered, he can enter the seller role at his own discretion. Ideally, the system should be able to confirm that the user's vehicle is indeed parked nearby. As soon as the sale begins, the seller's location is continuously transmitted to the system. The seller needs to provide the following information:

- Location of the spot
- Description of the user's vehicle
- In how much time the spot will be free
- How long the user commits to wait for a buyer

Location The location is retrieved automatically from the phone. There is a need for filtering the accepted spots, as the focus of the application is urban areas. The system should reject remote locations where there is little chance to find a buyer so as to prevent sellers gaining credits in an abusive manner.

Additionally, having a pricing strategy depending on the location of the spot is one of the possible improvement of the prototype, and would be in line with the dynamic pricing option explored in the first part.

Vehicle Description This description includes vehicle brand, model and exterior color as well as an optional text description for the user's to include distinctive traits he wants to share. This description is needed from the buyer's perspective to identify, hopefully uniquely, the spot in a street or parking zone as location cannot be accurate or reliable enough. Alternatively, a photo of the car could be used as description.

Buffer Time This parameter allows the seller to plan his departure in advance and represents valuable information for the system as the spot can then be offered immediately to buyers who are not in the immediate vicinity but who are projected to spend that amount

of time driving to the spot from their current location. This parameter can of course be set to zero to signal that the seller is ready to leave immediately. Conversely, there is a need for a reasonable upper limit to prevent abuse and erroneous inputs.

Wait Time This is the time the seller is willing to wait while ready to leave at a moment's notice. This parameter will be used to calculate how far potential buyers can be: the shorter time the seller is ready to wait, the nearer buyers need to be. The rationale being that the system does not direct buyers to sellers that are not ready to leave in order to avoid buyers having to resort to illegal or double-parking. Sellers are incentivized to wait longer as this parameter crucially determines the availability of parking spots within the community. In total, the parking spot will be available to buyers at most $T_O = T_W + T_B$, where T_O is named the *offer time*.

After submitting these details, the user is considered as an active seller by the system. At this time, the seller can retract without consequences. During Wait Time, as well as during an exchange, the user is required to be geographically near the parking spot, leaving the vicinity will trigger a cancellation of the sale at the seller's expense.

Internally, the system will trigger a search for an active buyer located near enough to drive to the spot within the time constraint chosen by the seller, that is buffer time plus wait time. If no such buyer is found when sale time runs out, the seller is credited with a certain amount of credit and then exits the seller state.

If the system finds a suitable buyer that accepts the exchange, the seller is notified that a transaction is taking place. The seller is given information on the distance between him and the buyer as well as the estimated time of arrival. The seller can also locate the buyer on the map in real time.

When buyer and seller are in visual contact with each other, the seller is allowed to vacate the spot leaving it free for the buyer. Close proximity is validated through a correlation of the current location of both parties. After leaving the spot, the seller is asked for feedback in the form of a numerical score. An optional text comment could be useful as well if the user wants to communicate a specific opinion. The seller also has the option to cancel the exchange at any time at his own cost. When doing so, he will be asked to choose from a list of predefined reasons.

If the buyer cancels the exchange, the seller is notified and the system will start searching for a new buyer within the new time constraints. After both users provide feedback, the exchange is closed and the seller leaves the seller state, earning credits. The cost and gain of credits resulting from an exchange is determined by the feedback of both parties

and is the subject of the following section.

Buyer's perspective

The user can enter the buyer state at any time except when in the seller state, as the roles of buyer and seller are mutually exclusive. Again, the system should be able to confirm the user is driving though activity detection. After a user makes the request to buy a spot, the system will trigger a search among available sellers.

If a suitable seller is found, the buyer is notified and has access to some information about the spot's location in order to decide for himself if it is adequate. The buyer can only see the distance he would have to drive as well as an estimated time of arrival. The buyer's acknowledgement of the proposed exchange represents a commitment to the system since the spot cannot be offered to other buyers after this stage. Obviously, buyers cannot have access to the precise seller's location before making this commitment. The buyer can cancel its request for a spot without penalties as long as he did not accept an exchange, if he finds a parking spot independently for example.

If the buyer refuses an exchange, the system will search for another suitable seller until the buyers accept an exchange or cancels his request. When the buyer accepts an exchange, he is guided to the precise location of the seller.

At this point, the buyer can still cancel the exchange, but a penalty will incur as he did not stick to his commitment to buy the spot. This is called "breaking" the exchange.

When the buyer arrives on location, buyer and seller make use of each others vehicle's description to precisely locate each other. The seller has the responsibility to only leave the spot when reasonably sure that the buyer will be able to take it. When the buyer signals a successful exchange, he is asked for feedback in the form of a numerical score. If the exchange is unsuccessful, the buyer is asked to give a reason, and the system will start searching for a new seller.

If the buyer takes too much time to arrive to the location and the seller decides to leave, the buyer is notified that the exchange could not be completed due to seller's departure. The buyer has the option to continue following guidance to the spot, without guarantee that the spot will still be free. The second option is to search for a new seller.

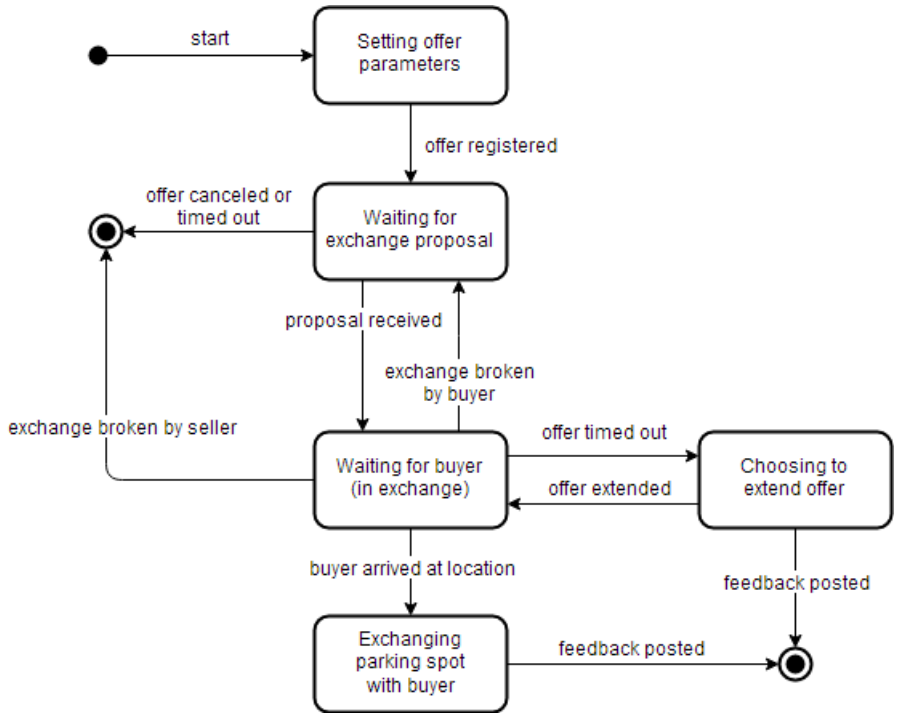


FIGURE 6.1: Informal seller state diagram

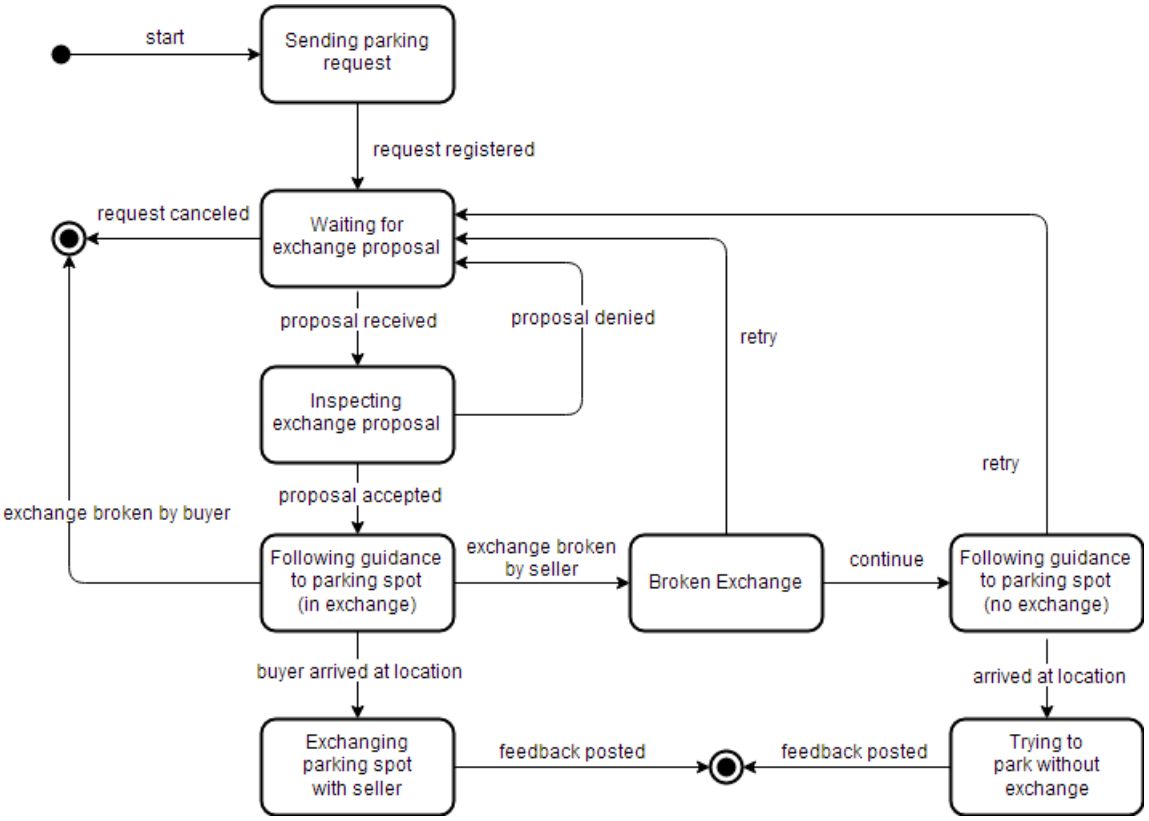


FIGURE 6.2: Informal buyer state diagram

6.2.2 Virtual Economy

Building a functional economic for our trade system is not easy, as we made clear in the previous part of this document. Many parameters need to be taken into account, and safeguards against scams and untrustworthy users have to be applied in every step of the trade to ensure the satisfaction of our honest users. We will now go more into the details of how we designed our economic system, and how the amount of points required to buy and sell spots have an impact on our users motivations and willingness to accomplish successful trades.

In the application, those points are called “StreetCreds”. Every purchase of a parking spot costs a predetermined number of “StreetCreds” to the buyer, and the seller on the other hand receives “StreetCreds” for having successfully sold his spot. We call C the number of credits spent by the buyer, and G the number of credits received by the seller.

Having G and C, we also define N as the credits of a newly created account. We already discover that with currency being constantly added by the creation of new accounts, we have to ensure $C > G$ most of the time. Without diving to much into the final values, we also have to relate N to the two others parameters. We want our new users to be able to buy a spot with their initial credits, as a mean to discover and test the service, so we have to ensure $N >= C$. Since there is only one way for a user to generate currency, we will set N up so a new user can buy two trades, and even have some “StreetCreds” leftover. The consensus we ended up after some reflexions between ourselves and CommuniThings is $N = 2.75 * C$, so the user has enough credits to purchase 2 spots before needing to sell a spot himself, giving him the opportunity to test the service and realize its advantages, but also isn’t left with zero points after the two trades. This is done intentionally as a mean to discourage users to simply create a new account since he would be "wasting" the remaining points on his account, that are close to allow him to buy another spot, since he is left with $0.75 * C$ “StreetCreds”.

There is at least one more value we need to consider. When we introduced the concept of user’s reputation in the Community chapter, we mentioned contested trades. Those happen when the seller and the buyer disagree about the outcome of the trade. It can happen when one of the two tries to break out of the trade but doesn’t want to assume the responsibility and loose its rewards, so he uses the feedback system to try to dodge the loss, or in some cases (the Trolls we talked about in the Community chapter), simply to annoy the other user and make him lose points. In theses scenario, we do not want the user that stayed honest to lose all of the rewards he deserves, but we cannot always be sure of which party lied, so we have to introduce a refund value to give to both users in case of contested trade, when the dishonest party cannot be identified.

In designing our economy, we will have to solve the issues of false sellers, or farmers, i.e. people who would try to sell false spot to earn points quickly, dishonest buyers, who would claim that the trade wasn't successful to keep their credits and the opposite, dishonest sellers who would try to call off the exchange to either leave sooner or simply annoy the buyer.

Since we are not the first to develop such system, we can obviously use what our predecessors have worked on. We already mentioned CrowdPark [38], a system similar in many ways to what we present, also using a point-based system for trading. They presented a system that proposes a solution to some of the issues we mentioned. For the problem of the dishonest buyer, they propose a system where the points gained by re-selling the spot successfully are greater than the refund obtained by being dishonest. However, a buyer that would lie about the result of the trade and thus say that he didn't get the spot when in fact he did, cannot re-sell his spot since he never occupied it in the first place according to the system.

For example, if selling a spot gives the seller 10 points, and buying a spot requires 12 points, while the refund for a contested trade is 4 points[1], then by implementing a bonus of 5 points when re-selling a spot previously bought, the user has a net gain of 1 point if he stays honest and then re-sell the spot, which accomplishes two goals, first discouraging the buyers to lie, and also encouraging the buyer to sell the spot again thus creating more trades.

With this in mind, we also have to consider the issue of two users trading a spot back and forth to "farm" points, since with this method in usage, the first trade would result in a +10 points for the first user, and -12 points for the second one, this making them loose point which makes farming not profitable. But after the second trade, if we consider that they're allowed to get the re-sell bonus, the gain/loss becomes -2 for the first user (+10 first trade, -12 second trade) and +3 for the second user (-12 first trade, +15 second trade with the re-sell bonus), resulting in a +1 net point gained between the two of them. If we keep going, after the third trade we have a +4 points gained (user 1 has gained 13 points overall, user 2 has lost 9 points overall), each consecutive trade making them earn 3 points globally, both user having a positive balance after the fourth trade.

[1]Reminder : we call a trade contested when during feedback either the seller or the buyer claims that the trade was unsuccessful while the other party claims it was successful.

Exchange #	User A balance	User B balance	Net gain
0	0	0	0
1	10	-12	-2
2	-2	3	1
3	13	-9	4
4	1	6	7
5	16	-6	10

Therefore, with this system in place, farming point can effectively be done, the rate and effectiveness depending on the spot trading value. But it is a very repetitive process which can either be slowed down and/or detected. One easy way to reduce the farming efficiency would be to set a time limit between two trade a user can do. If we set a 15 minutes period during which a user cannot start another trade, since we consider that most of the time a user leaving his place won't park that soon, and a user getting the place will be using it for at least that time, getting to the fourth trade takes 45 minutes, making it a lengthy process. The other option to slow it down would be to reduce the re-sell bonus, but in turn we would also have to reduce the refund for contested trade, making it really low and giving more power of annoyance to dishonest seller or buyers.

But we can also detect it, since it takes 4 trades to obtain a result that is productive for both users, we can implement a safeguard in our system that detect if two users engages more than two trades in a certain timeframe, they could get a warning and ultimately a ban if they persist trying to abuse the system. The way the community and the trade algorithm works, it is indeed very unlikely for two persons to be matched two times in a trade naturally (as of without manipulation from either side) within the same hour.

Cost-Benefit analysis

In total, we distinguish between 7 possible outcomes to a parking spot offer:

1. No exchange took place: the seller waited but no buyer was found.
2. Unsuccessful exchange: The seller broke the contract and left early.
3. Unsuccessful exchange: The buyer broke the contract and never arrived.
4. Unsuccessful exchange: Both parties report a failure.
5. Contested exchange: The buyer reports success while the seller reports failure.
6. Contested exchange: The seller reports success while the buyer reports failure
7. Successful exchange: Both parties report the exchange as successful.

No Exchange took place: The most straightforward possibility, but also the less desirable, is that the system could not find a suitable buyer for the parking spot. In this case, the seller is credited with an amount of credit G based on the duration of the offer, specifically on *active time* and *inactive time*:

$$G(T_b, T_w) = K_1.T_b + K_2.T_w$$

where K_1 and K_2 are positive constants such that $K_2 > K_1$. While the *buffer time* could be considered "dead time" as no exchange can take place during that time, it still brings valuable knowledge into the system and as such, sellers should be incentivized accordingly to sell their spots in advance. While no exchange is taking place, the seller can cancel his offer at any time without penalties, gaining G credits according to the total time the seller actually waited, as opposed to the time the seller originally set. Specifically, the seller will gain 0 "StreetCreds" if he cancels his offer while still in buffer time and $G(T_b, T_(now))$ "StreetCreds" if in wait time.

From the buyer's perspective, there is no cost or gain associated with canceling a request if no exchange is engaged. To terminate a request in the absence of sellers, the buyer has to cancel it manually or it will eventually timeout.

This model could be expanded to take into account dynamic parameters such as the demand for parking at this location at this time in the similar manner to Uber's surge pricing.

The seller broke the exchange: If the seller chooses to break an active exchange, he will be credited with either 0 or $G(T_b, T_(now))$ "StreetCreds" where $T_(now)$ is the time the seller waited, ready to leave, until he broke the exchange. But the seller will also suffer a penalty of $P(D) = K_4.D$ credits where D is the distance between the seller and the buyer's current location. Therefore, the seller is encouraged not to break the exchange when the buyer is nearby.

If the exchange exceeds the seller's *offer time*, the seller has the opportunity to break the exchange at no cost, then gaining $G(T_b, T_w)$ credits without penalties. Alternatively, the seller can set a new *wait time* of N minutes at his own discretion. At the end of this period, the seller will gain $G(0, N)$ "StreetCreds" and will be asked to continue again if the exchange is still not fulfilled. However, if the seller breaks the exchange before the new *wait time* runs out, he will suffer a penalty of $P(D)$.

On the other hand, the buyer does not lose any credits if this occurs, he will simply be notified and the system will suggest another seller. However, if the buyer decides to cancel his request for parking, he will still lose the credits he invested when he made the request in the first place.

The buyer broke the exchange: When the buyer breaks an active exchange, he loses all the credits he invested. The next time the buyer wants to use the system to find a parking spot, he will have to spend C credits again.

The seller is notified that the exchange is canceled but does not suffer any consequences. If the seller's *offer time* allows it, the system will search for a new buyer.

The Exchange is contested by the seller: If the buyer reports the exchange as a success, then the system fulfilled its main mission. Therefore, the buyer should not suffer negative consequences in the event he found a free parking spot on his way to the seller's location for example. Furthermore, we have to take into account the possibility that a seller gives negative feedback with malicious intent. The cost C paid by the buyer is still debited from his account.

The seller continues to offer his spot unless his *offer period* is too short, in which case the offer is closed and the seller gains G credit.

The Exchange is contested by the buyer: In this case, the seller reports a successful trade while the buyer disagrees. This combination can be interpreted in multiple ways. Maybe the seller left early but did not want to endorse his responsibility by breaking the exchange or the buyer wants to cause harm to the seller. In this situation we decide to penalize both users: The seller will gain $G/3$ "StreetCreds" while the buyer loses $2 * C/3$ "StreetCreds". While it may seem advantageous for the buyer to systematically report a failed exchange as it costs less, the buyer will not be able to benefit from the resale bonus. It is then necessary to ensure that the resale bonus R is greater than $C/3$.

Both users report a failure In case both users report a failed exchange, we can reasonably infer that the exchange effectively could not be completed, unless the 2 users collaborating in the aim to exploit the system. Therefore, this situation should cost both users as well. With this logic in mind, we apply the same parameters as for the previous case.

The Exchange was successful: In the case of a successful exchange, the seller gains the same amount of credit $G(T_b, T_w)$ as if no exchange took place regardless of the time at which the exchange is done. This way, while there is no explicit bonus for completing an exchange for the seller, he is rewarded gaining the full amount for which he committed. The buyer spends a fixed amount of credit $C = K_3$ where K_3 is a constant, when he sends the request for a parking spot to the system. This value C must be carefully chosen in regards to various parameters of the virtual economy such as the total amount of credits in circulation and inflation.

It would also be possible to integrate dynamic pricing on C in regards to the particular location and time of day.

In summary:

	Seller (gain)	Buyer (cost)
Request canceled or timed out	$\varnothing$	0
Offer canceled	$0 \mid G(T_b, T_{now})$	$\varnothing$
Offer timed out	$G(T_b, T_w)$	$\varnothing$
Exchange broken by Seller	$G(T_b, T_{now})$	0
Exchange broken by Buyer	0	C
Exchange contested by Seller	0	0
Exchange contested by Buyer	$2 * G(T_b, T_w)/3$	$2 * C/3$
Exchange unsuccessful	$2 * G(T_b, T_w)/3$	$2 * C/3$
Exchange successful	$G(T_b, T_w)$	C

6.2.3 User reputation score

We introduced in the first part the notion of user reputation, as a tool used by games and communities to handle Trolls and untruthful users. For example, in League of Legends (the game we spoke about in the part one), there is a clear benefit for being of good reputation such as in game rewards, and there also is a broader range of sanctions Riot Games can inflict on their players without having to apply a ban. In our case, the sanctions we can inflict are rather limited. We have of course the option to ban a user, but it would be best to apply this only on extreme cases. Another option would be penalties on user's points if his reputation score drops below a certain threshold, and lastly we could forbid him from buying spot, only making him able to sell his spot. This solution could however prove to be dangerous as it could tempt the user to simply "troll" again, out of frustration, and lower even more his reputation score.

Our best course of action is then to have the user's reputation impact what they mostly want, their ability to find a parking spot. What we implemented is to have a reputation score, an integer value, that start with a default value D. During the matching algorithm, the user reputation is then taken into account to determine which user in the area gets the parking spot sold, a higher reputation leading to better chances of getting the spot. This is advertised frequently, most notably when the feedback and score of a trade is asked to the owner.

We also decided not to use a "Box" system, where users only interact and trade with users of similar reputation, as this has been proven in multiple cases to be great for high reputation users, but ultimately extremely negative for users dropping in a lower class of

users and trading more frequently to unreliable users, leading to a negative spiral.

The factors influencing the reputation of a user are limited. We determined that the reputation had to increase when the user was completing a trade successfully, or in the case of failed trade declared such by both parties, and breaking a trade. Those cases distinguish themselves of the others possibilities (the trade outcomes specified right above this section), as they are situations where our user, most likely, both tell the truth, and thus are in a way awarded for this, even if the trade failed. Also when breaking a trade, you lose your points but are rewarded for your honesty.

The other two outcomes are the contested trades. We already detailed them in the economic section. Without certain knowledge of the lying user, we choose the option to lower the reputation of both. This makes the contested trade the least desirable outcome in terms of reputation, giving our users more incentives to not trigger them.

Each time a user is about to perform an action possibly lowering or increasing his reputation, a warning message announces him the amount of points won or loss by the action, and that could impact his priority getting spots. When dropping below a certain threshold, he is notified that his reputation has gone too low, and that he should try to become a more reliable member of the community. If the users drops below a critical threshold, his account is suspended.

6.2.4 Matching Algorithm

As presented in the previous section, buyers and sellers are matched by the system. From the buyer's perspective, the choice of the best parking available is realized as a series of exchange proposal that the user can either accept or deny. This automatic matching frees the user of having to consider multiple options at once and answering the proposal is straightforward. As the buyer could be on the road driving, it seems appropriate to ask for as little attention from the user as possible. Buyers can refuse as many proposal as they want without penalties, but the offers that were denied will not be suggested again for the entire duration of the buyer's request. This way, users are motivated to accept proposals sooner instead of gambling on the next proposal to be better than the current one, which would slow down the matching rate between sellers and buyers in the system as a whole. It is indeed desirable that an exchange can be created as soon as possible as both request and offer are short-lived.

When the system receives a parking request from a buyer, there is no particular process started by the system. The request is simply registered in a pool holding all the requests in a particular region. The match algorithm is initiated when a parking offer is

received from a seller. The system will then query the request pool for the most appropriate parking request. The matching criteria for a suitable request are the following:

- **Vehicle size** : The seller's vehicle must be at least as large as the buyer's vehicle, in order to avoid the situation where the buyer cannot physically park on the seller's spot. A buyer driving a small city car could match with a seller driving a big SUV for example, but not the opposite since the vehicle size is the only reliable way for the system to estimate the size of the parking spot.

- **Distance** : Obviously, buyer and seller have to be near each other for the exchange to be of interest. Requests and offers registered in the system have joined GPS coordinates, on which the distance computation is based on. Shortest distances are prioritized, with a hard ceiling based on the seller's *offer time*. The *offer time* may be translated to distance in various ways, the most basic being a simple multiplication by an estimated average vehicle speed in the city: $D = T_O * S_{avg}$. More complicated models may be used to take traffic into account, either by the use of statistics or real-time data. It is interesting to note that this park sharing application combined with activity detection could be used to crowdsource traffic data in a very streamlined fashion, however we consider this being outside the scope of this master's thesis.

- **User Reputation** : Finally, the user's reputation is also a parameter in the match algorithm. While there should be no "boxing" of users as mentioned in chapter 1, reputation may be used as a final selection to pick one request among all the requests that matched the 2 previous criteria: the request from the buyer with the highest reputation will be selected first.

If a suitable request is found by the match algorithm, a notification is sent to the buyer. The seller is notified when and only if the buyer accepts the exchange. If the buyer refuses, the system flags this particular couple of requests as prohibited and cannot be matched again.

If no suitable request could be found due to a lack of demand, the process will pause for a short time then starts again, in the hope that the request pool now contains at least one suitable request.

Algorithm 1 shows a description of the algorithm in pseudo-code, assuming that the properties of offers and requests are updated in real-time by another process in parallel.

As time moves forward, the value of *d_min* and *d_max* will change. Figure 6.3 shows the pattern of concentric circles that appears if we assume that the algorithm waits 4 times in the while loop.

Algorithm 1 Request-Offer match algorithm

```
1:  function THRESHOLD(latitude,longitude,time)
2:      speed_avg ← CONSTANT
3:      traffic_factor ← Traffic_Estimator(latitude,longitude,now)
4:      return speed_avg * traffic_factor * time
5:  function MATCH(seller,offer,retry_timeout)
6:      request ← NULL
7:      while request is NULL && offer.canceled is FALSE do
8:          distance_min ← THRESHOLD(offer.latitute, offer.longitute, offer.buffer_time)
9:          distance_max ← THRESHOLD(offer.latitute, offer.longitute, offer.buffer_time+offer.wait_time)
10:         M ← query_pool(d_min,d_max,offer.vehicle_size)
11:         if M is not empty then
12:             request ← highest_reputation(M)
13:         else
14:             wait(retry_timeout)
15:     return request
```

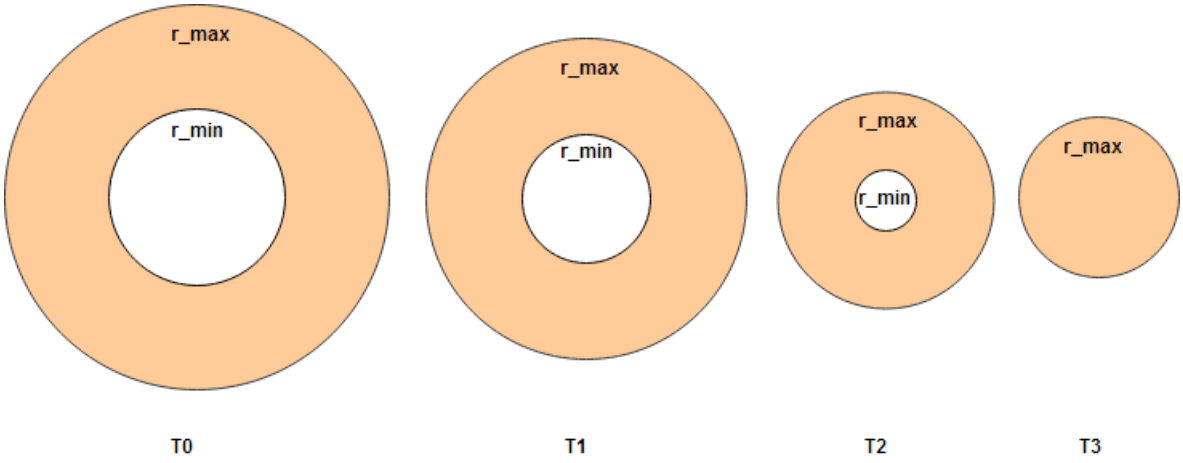


FIGURE 6.3: Maximum and minimum search radii as time evolves

6.2.5 Activity Detection and Ephemeral Parking

The model presented so far relies almost exclusively on economic incentives and user honesty to deter nefarious behaviors. For example, a user could decide to illegitimately sell a parking spot that he does not occupy then accept an exchange with a buyer in the intention to waste the buyer's time. While this behavior will result in the seller losing "StreetCreds" and reputation, if the seller does not care about those things, the only solution would be to ban him after a number of offenses. It would be highly desirable to put in place some other counter measures that would substantiate the user's claims at the beginning, rather than having no other choices but to trust the user, let the exchange proceed and witness a failure afterwards.

There are two key points to consider. The first is for a user to endorse the role of buyer, he must be actively driving, or the request for parking would not make sense. The

second key point is that the seller's car must have been parked recently and the seller is currently not driving. Both of these conditions could be checked through activity detection techniques. If the system can enforce that users are performing a certain activity before accepting a request/offer, a whole category of toxic behaviors can be avoided as a result.

This is not the only benefit of activity detection however. As mentioned in section "parksharing at larger scales", an activity monitor module could also be used to detect when a user leaves a parking spot without the need for user interaction. Combined with a park sharing model similar to the one adopted by ParkTag [28], it could prove to be a powerful tool in smoothing traffic in urban centers. While we believe the complete, end-to-end, exchange model we discuss here has numerous advantages over this kind of "first come first served" approach, it is a heavy process whose benefits can only outweigh the inconvenience in high demand situations. Therefore, we also believe a combination of the two would be the most effective solution.

Fortunately, our exchange model lends itself well to this combination. As selling a spot requires user initiative, we naturally assume that some users will forget to initiate the process or deliberately avoid to because they are in a hurry. In this situation, potential buyers would still have a benefit in being informed of the spot's location, even if there is no guarantee about its availability when they get there.

With this in mind, we suggest the concept of "Ephemeral Parking". When a free spot is detected through activity detection, without user involvement, the system will create a temporary free spot that is visible to all members of the community for a limited period. This adds a second layer to our park-sharing application: Buyers have the option to wait on their own until a nearby ephemeral parking spot appears on their map.

Chapter 7

Technologies

7.1 Mobile Framework

7.1.1 Overview

Choosing the right framework is a critical decision in the application requirement analysis and planning. In recent years, multiples mobile framework have emerged on the market, providing support for the development of cross-platforms applications. Amongst theses frameworks, two of the most prominent are Ionic, developed by Drifty, NativeScript, developped by Telerik and Xamarin, recently acquired by Microsoft. Of course, there is also an argument regarding the use or not of such framework, as developing applications directly with Native code is always an option that presents some specific advantages.

In a normal situation, we would have assessed the performances, perks and issues with each framework before starting our work, most likely by doing an easy test application to compare each one. But in our case, since we worked with CommuniThings, who was already developing applications that we had to interact with, if not for a complete merge in the end, at least for some conjunction between them, the choice was not ours, we had to work with NativeScript. But this does not prevent us from comparing and judging the option in order to determine if this was the optimal framework for us, or if, in other circumstances, we would have chosen another one.

7.1.2 NativeScript

First, let's start with NativeScript then [1]. It is an opensource hybrid framework. NativeScript primary goal is to be perfectly hybrid between iOS and Android, meaning having your code to be exactly identical when switching platforms. On the code side, it allows either JavaScript or TypeScript. While using Javascript, the UI pages has to be written in XML, while the usage of HTML is required when using Typescript. TypeScript uses the AngularJS framework which simplify the links between UI components and their programmatic behavior. With the use of XML or HTML tags for the UI, it calls the correspond component when compiling the code. So for example if you use a "NativeScript-button"

in your UI page, compiling the code on Android will translate in a Android Button, while compiling on iOS will translate to a iOS UIButton.

NativeScript also let you access the natives API of both operating systems, which means that even if some components (sensors like the accelerometer for example) are not reachable through the standard NativeScript API, you can access them directly through the Android API, but losing some adaptability in the process. Telerik is reactive and the framework is regularly being updated, with a community publishing a lot of module offering more functionality that are initially available. But as it offers an interesting diversity, it is also one of the framework drawback, since you have to search for, and install every module independently and manually, and add them yourself in the dependency list of the application. Another issue with NativeScript is the application size, quite higher than its concurrent. We also have to mention the documentation, rather unclear and sometimes incomplete. Being allowed to code in either Typescript or Javascript causes some annoying situations, some modules are clearly not designed to be used in either TS or JS and the examples given by Telerik sometimes lack one of the programming language. In terms of performances, it also causes some loss compared to direct native call when using plugins.

In term of speed of development, NativeScript allows for a fast development cycle, changes applied to the code do not require the applications to be rebuild and thus can be applied, tested and debugged instantly on either a physical or virtual device.

7.1.3 Native

Of course it is also possible to write your code directly using the natives Android and iOS APIs, respectively using Java and Swift. Doing so, you have access to basically everything directly, without the use of modules or dependencies, but you'll have to code the application twice, if you want to make it available on both operating systems. So basically, in comparison when not using a framework, you gain control over the code and the exact implementation of the application and the UI design, but you loose in adaptability and efficiency, since you will be doing the same work multiple time to adapt your system to different APIs, with an additional risk of having some options available on some OS but not on the other one.

By going purely native, performance is the biggest advantage, as you can access every component directly through the corresponding API when other frameworks requires the usage of plugins, that might not always be perfectly optimized. One of the other advantages is that you have access to very complete documentations from Google and Apple and multiples examples of applications from people having developed apps natively,

while some framework, being younger, might sometimes lacks examples and online support.

7.1.4 Xamarin

Xamarin is a framework initally developped by Ximian, and was acquired by Microsoft in February 2016. Since the acquisition, Xamarin has been made open-source and free to use, as it is now included in the Community Edition of Visual Studio. However, the Professional License for Visual Studio, that is recommended for building large scale applications is still requiring a license. Xamarin is based on the .NET framework, and uses C#. It is close to being hybrid, but not totally, meaning some of the code will need alteration if the developer wants to create both an iOS and an Android application. This is the consequence of Xamarin code being really close to native code in regard of component access, when NativeScript and Ionic often requires plugins for such functionalities. This also means that performance-wise, Xamarin is way ahead of the hybrid frameworks and is close to what a pure native application could offer.

The portability issue is also quite important for the UI, as its code will have to be mostly platform dependent. This can cause complications and a lot of additional work for applications where the UI part is critical and a big part of the application. It is also important to note that Xamarin development is slower than for the Hybrid frameworks, as every changes in the code means that you have to completely re-build the application.

Regarding the documentation and support, having Microsoft behind is obviously making everything safe, even if the documentation for Xamarin in itself is sometimes lacking some informations.

7.1.5 Ionic2

Ionic2 is really similar to NativeScript. They are both opensource and hybrid. It allows the usage of TypeScript and Javascript, even though the framework clearly encourage the usage of the former and with it the AngularJS framework. It relies also on plugins to access the native Android or iOS functionalities. However, it seems every plugins wraps callbacks in Promise or Observable, making the usage of such plugins really heavy if the developer happen to require a lot of them, and a plugin is needed for each native functionality the programmer wants to use.

This is the main difference between NativeScript and Ionic. As said previously, Nativescript allows for direct access to the Android and iOS API, while Ionic will require

plugins, and thus a Promise callback most of the time. Other than that, the difference between the 2 frameworks are really marginal, beside documentations, Ionic being mostly TypeScript nearly all the documentations is oriented with this langage in mind, while NativeScript plugins are mixed, having some plugins written for JavaScript while others are in TypeScript, knowing that conversion from one language to the other is not always straightforward.

7.1.6 Overall

In our case, having prior knowledge of C#, Java and Javascript, but no extended experience with any of these languages except Java, going fully native would have been the safest choice in term of time invested in learning the technology or framework. But being also limited in manpower, going native alongside a goal of developing for both iOS and Android would have been to much of an issue for us, if the choice of the framework had been ours.

Xamarin would also probably not been wise, as our application has some complex UI components, and combined to the learning we would have been required to use Xamarin, loosing probably lot of time. Additionally, the cost for the professional license itself makes it difficult to consider Xamarin seriously, in our case.

With this is mind, using a hybrid JavaScript/TypeScript framework seems the right option. Javascript does not require a lot of learning, the UI components being HTML or XML means we can handle them easily. The fast development cycle is also beneficial since the application we are developing is not, as of now, a long term project and had to be operational as fast as possible.

We can say that having to work with NativeScript is not a bad choice, and is most likely our safest bet for the applications we had to develop. We could have used Ionic2, but the differences between the two seems so marginal that it basically comes down to a choice with no real impact on the application development.

7.2 FIWARE

The FIWARE community is a European based open community that aims to accelerate the development and deployment of innovative applications, with a focus on Smart City services, Media and content as well as E-Learning. The community supports various active programs to achieve its stated goal “To build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease

the development of new Smart Applications in multiple sectors."[1] The three main programs:

FIWARE Platform

The FIWARE Platform is a set of open-source specifications for API's that are called Generic Enablers, or GE within the community. These GE are designed to be general purpose, hence the term, with each GE offering a set of functionality to address common tasks in large-scale application. FIWARE also specifies how these GE interact with one another so they can be easily combined. An implementation of a GE is referred as a Generic Enable Implementation or GEi. Multiple GEi in various development stage might exists for a single GE, they are made available openly through the FIWARE Catalog.

FIWARE Lab

FIWARE Lab is a sandbox environment in which users can experiment with FIWARE technologies at very little or no cost. Users can book instances of several GEi's to integrate with their existing infrastructure and evaluate how they can be leveraged. FIWARE Lab also publishes Open Data from cities and other organizations for users to experiment with.

FIWARE Accelerate

The Accelerate program aims to promote the adoption of FIWARE technologies. The program focuses primarily on small business and start-ups, to which it grants funding through an EU campaign for innovation called "Horizon 2020".

7.2.1 Publish/Subscribe Context Broker GE

The FIWARE Publish/Subscribe Context Broker General Enabler is a specification for a specialized database called a Context Broker. This GE acts as a repository for contextual data produced by entities referred to as Context Producers. Contextual data may then be processed by other entities called Context Consumers. Both producers and consumers may be applications or other FIWARE GE. Change in contextual data is encapsulated in events that are notified to subscribed context consumers. The interface supports both push and pull models from/to both producers and consumers.

The fundamental principle at play for this GE is the maximum decoupling of producers and consumers, with the context broker acting as a middle layer. This model makes it simpler to deploy data collecting devices at large scales since they do not need to be directly connected to the services that need the data they generate. Context producers can

[1] https://www.fiware.org/about-us/

publish their data using simple logic, knowing that the context broker will reflect their past and current state independently. Context consumers can then use the context broker as a centralized access point to numerous and various IoT devices. Crucially, contextual data generated by the producers is query-able using a custom "Simple Query Language", allowing consumer applications to aggregate data in a transparent way, completely abstracting away the data source.

7.2.2 Orion Context Broker

CommuniThings uses an implementation of the Publish/Subscribe Context Broker GE named Orion as a component in one of their application. Consequently, we conducted a review of this GE and concluded that it could be useful for the ParkExchange project as well. Particularly, the ability to perform geolocation based filtering on queries to the context broker is a very desirable feature, as the application will have to match users with other nearby users. Performing this match with Orion is simple and efficient according to the specifications [2].

Moreover, the ability to distribute context producers and consumers across multiple independent services could prove to be a straightforward way to extend the ParkExchange service to multiple cities in the future [3].

[2] http://fiware-orion.readthedocs.io/en/master/user/geolocation/
[3] http://fiware-orion.readthedocs.io/en/master/user/service_path/

7.3 Laravel

Building a codebase from scratch, while offering greater flexibility, would require exhaustive efforts and is certainly not recommended for a proof of concept. Thus, in order to speed up development time and maintainability, using an efficient framework is crucial.

At Communithings, all backend development is done in php and, more recently, is powered by Laravel. Since the main goal of our proof of concept is to reuse some components from CommuniThings, we chose to use Laravel as well in order to facilitate the integration of ParkExchange in the existing codebase of CommuniThings. This way, we also hope to make it easier for CommuniThings developers to actually use the code we developed in the future as a base for a full-fledged park sharing application.

Fortunately, Laravel offers interesting features for our case study and does not inder productivity or pose any particular challenge.

7.3.1 Artisan CLI

Laravel ships with a command line interface named Artisan. This tool is very useful for developers, it can notably be used to deploy the application on a lightweight local server, manage the database in conjunction with Eloquent models and interact directly with the application at runtime by inspecting or updating php objects.
Artisan dramatically speeds up development by making deployment and testing really fast.

7.3.2 Eloquent ORM

Eloquent ORM makes it easy to implement a data access layer. Eloquent bridges the gap between database and runtime application. Every database entity is encapsulated in an Eloquent Model that can then be used in code to query and update the database. The main advantage of Eloquent is that it eliminates entirely the need to write SQL code, as everything is specified in php. Writing out the database schema is then trivial using Artisan.

7.3.3 Passport

Laravel Passport is an integrated OAuth2 server that is included in the framework. It makes it easy to secure web applications and APIs using the OAuth protocol. Token Management is mainly automatic. The main task left to the developer is to configure which urls require authentication.

7.3.4 Queues and Jobs

Queues and Jobs are a simple mechanic to handle asynchronous or time-consuming tasks in Laravel. It follows a standard delayed execution model where Queues represent threads that executes jobs according to some scheduling strategy. Particularly, it allows the implementation of "long lasting requests", requests made of a sequence of http requests, or that still need some form of active processing after the response is sent to the client.

Chapter 8

Implementation

8.1 Overview

Now that everything theoretical has been set in place, in is time to dive in the implementation of the prototype itself, and of all its components. In this chapter will we describe the architecture of the system we developed, give more insight on the interactions between those components, and explain in more details the work process of the mobile application prototype.

We will also explore several possibilities of improvement of the prototype, that usually refers to concepts explore in the first part, and that would increases the probability of such application to go mainstream on day.

8.2 Architecture

8.2.1 Big Picture

The solution involves 4 co-dependent components, communicating with each other through the web via the https protocol. The NativeScript clients can communicate with the Laravel web application through an API dedicated to the ParkExchange service. Client and server exchange messages in the JSON format. The dialogue is secured by the OAUTH2 protocol.

As previously explained, The ORION Context Broker is used mainly for its geo-based querying capabilities. Orion endorses the role of "request pool" mentioned in section 6.2.4: it holds entities that represent parking requests from buyers, updated in real-time by the client through the Laravel web application. The client first sends the request to Laravel in JSON format, then Laravel forwards the requests to ORION as a compatible NGSIv2 entity. When a parking offer is received by Laravel, it will perform successive geo-based queries to ORION in order to find a suitable request.

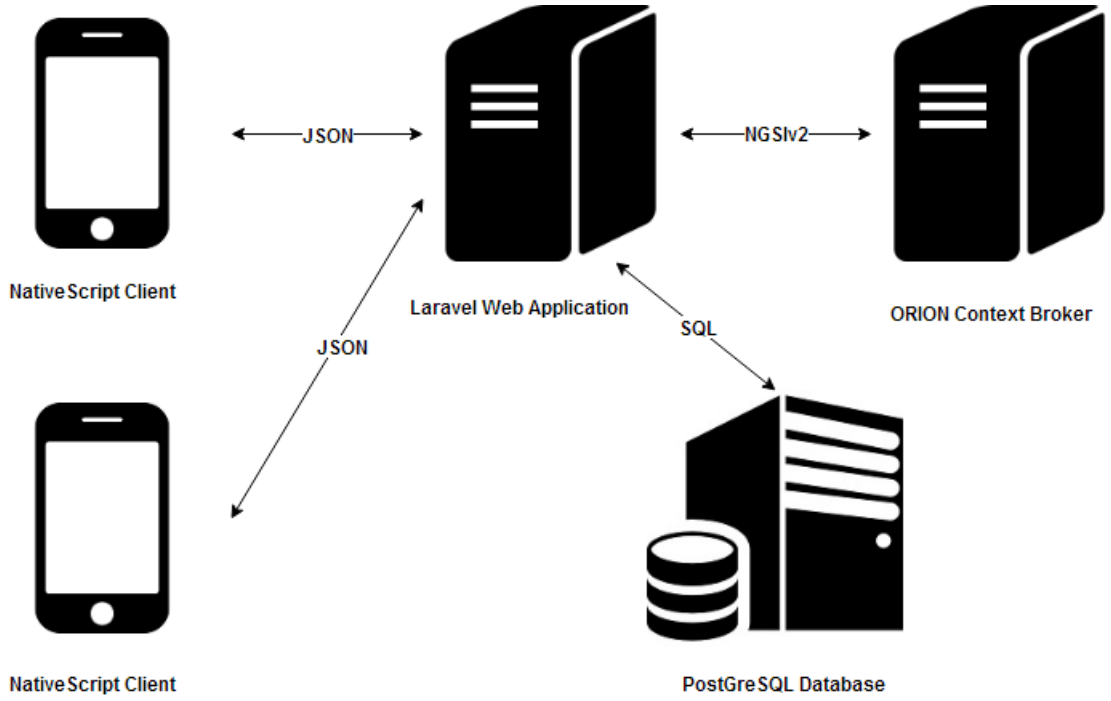


FIGURE 8.1: Simplified view of high-level architecture

The last component is a standard PostGreSQL Database that is the central data repository for the whole system. Entities inside ORION are transient, they are always assembled from Eloquent models who are saved in the SQL database.

8.2.2 SQL Database

The SQL Database is driven transparently from Laravel's built-in ORM: Eloquent. Every table in the database has a corresponding Eloquent Model that is used to interact with the database without the need for custom sql queries.

8.2.3 ORION Context Broker

There is no need for any custom development on the ORION side. Being a FIWARE Generic Enabler, the purpose of the context broker is to be usable as is by different applications for various purposes, and the ORION implementation certainly achieves this goal. We only need to define the properties of the entities we want to store in an ORION context, the rest of the mechanics involved is already built-in. We ave defined the parking request with the following attributes:

- **id** : a unique id for each entity is required by the NGSI standard. The id in SQL database is used as value.
- **type** : The NGSI standard also requires a type attribute. We set it to "ParkingRequest".
- **reputation** : the reputation of the user that sent this parking request.

- **vehicle_size** : an integer between 0 and 3 that represents the user's vehicle overall size. 0 denotes a small city car while 3 is used for large vans and minibus.
- **match** : the id of the offer that matched this request, or 0 if the request is not yet matched.
- **banned** : an array containing the idea of all offers that were denied by the buyer.
- **location** : an array that contains the current latitude and longitude of the buyer. It is updated continuously for the entire request life.

This set of attributes allows us to drive the search for a matching offer from the Laravel side, by using the Simplified Query Language of ORION.

8.2.4 Laravel Web Application

The web application we developed sits on top of the Laravel 5.4 Framework. We primarily followed the guidelines provided in Laravel's extensive documentation in terms of architecture.

We configured the Passport module in order to use the OAUTH2 protocol. A unique and secured API route group contains all the URLs necessary for the mobile application. The major part of the code sits in the controllers that process the HTTP requests coming from those URLs. We implemented two dedicated controllers to handle CRUD operations on users and vehicles respectively, and a third, larger, controller to handle all requests related to a parking exchange. The Exchange controller invokes static methods from a dedicated class to communicate with ORION, whose purpose is to encapsulate all NGSIv2 specific code in one module.

The Buyer/Seller Match algorithm presented in section 6.2.4 is implemented using Laravel's Queue mechanism. Every parking offer received results in the creation of a corresponding job that performs queries to ORION in order to find a matching request. If a match is found, the job will make appropriate changes in ORION and the SQL database so that users can be notified of the match when the Exchange controller answers the following requests, then gracefully exits. If no match can be found, the job will create a copy of itself and set it to fire with a delay, effectively doing the "wait" step of the match algorithm. Jobs are executed in parallel inside a separate process (the queue worker) so the web application itself remains responsive when the match algorithm is running.

8.2.5 NativeScript Client

NativeScript implements the MVVM pattern (Model – View – ViewController). For each page of the application, we created a HTML file serving as View, and a TypeScript file associated to the page for the ViewController. The model within the application is also done

in TypeScript. NativeScript provides the necessary module to easily binds each View to its ViewController, making the implementation of the pattern easier to neophytes. Each page, and modal view, is then a pair HTML-TypeScript, allowing to call function within the HTML page that are defined in the TS page, and the TS page contains a reference to the HTML page it is bound to.

We also defined a Singleton class "Global" containing our global variables, namely the current logged in user and with it the list of his available vehicle, and the address of the back-end server. Within this class are also defined several access functions to the global values.

Additionally, we also had to use what NativeScript calls a module page, that define every component available in the applications. Those can either be pages of the application, for which the component is then the ViewModel part (the typescript page) or plugins created by other members of the NativeScript community, documented online, that we decided to use for our application. This page is needed to be able to use major component in the other part of the application without having to declare them again. For example, by importing "NativeScriptHttpModule" of the "nativescript-angular/http" plugin, the application will automatically calls the NativeScript version of the HTTP request when such request are called in other components. Lastly, this page binds the page of the applications to their component name, allowing the routing based on said component name.

The routing itself is defined in another page, that simply binds a path name to component, for example linking the path "Login" to the component "LoginComponent", that was itself bind to the corresponding page and script in the module page. Thus when calling the function NavigateTo("Login") defined by NativeScript, or by calling links by their name in the HTML pages, the application can select which screen the user has to be send to.

Since we did not start from scratch but where we allowed to branch onto the existing project of CommuniThings, the application architecture, module and routing files were already existing and our task was then to understand their concept and utility, and to modify them to our own needs.

8.3 Mobile Application review

8.3.1 Map Screen

The first major screen of the application is the map. Most of it was designed by developers from CommuniThings, using MapBox and the corresponding NativeScript plugin. This screen is the main one of the application, it provides the users with the available spots of the Parking-Minute of the CT's application.

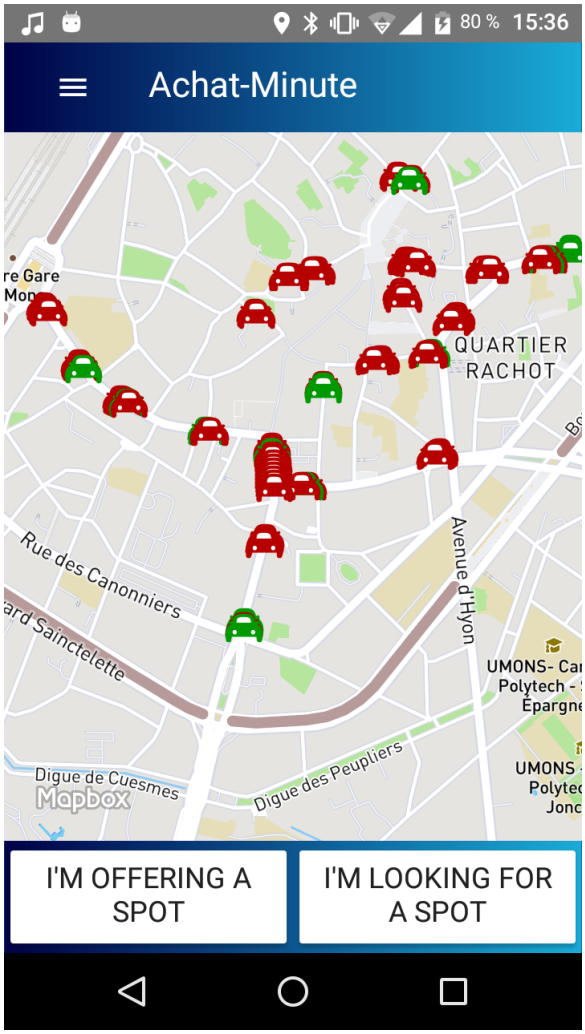


FIGURE 8.2: Prototype main screen

Our biggest contribution to this screen is the two button nicknamed "Sell" and "Buy" (respectively "I'm Offering a spot" and "I'm looking for a spot" on the UI). As their name indicates, their purpose is to initiate a trade by either selling a spot, or buying one. Both buttons trigger a modal view allowing the user to enter the options of the trade. For selling a spot, the user enters his buffer time and wait time, additionally to the car he is currently using. Depending of the input, the amount of StreetCreds gained by the user

is calculated. To buy a spot, the buyer simply has to indicate the car he is currently using. Both modals have a confirm button, needed to initiate the sell/buy (which is grayed if the buyer does not have enough points to buy a spot). When the search for a trade starts, the "Sell" and "Buy" button are modified. One is made inactive and displays an activity notifier, simply a looping symbol indicating that the application is still working, and the other button becomes the "Cancel" button, allowing the user to cancel the search.

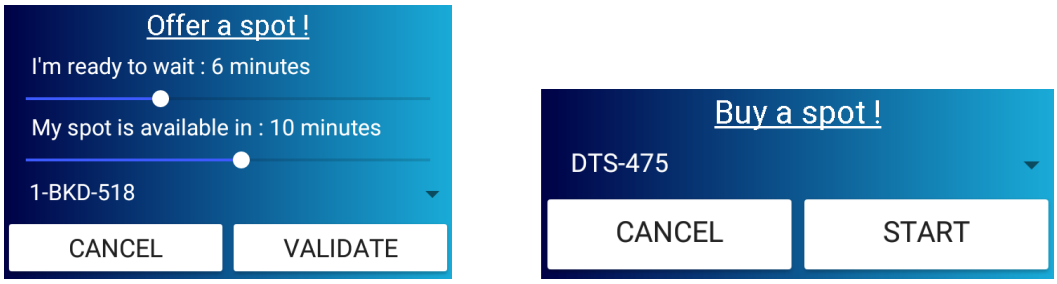


FIGURE 8.3: Sell and Buy modals

Buying a spot

When buying a spot, the application send a message to the backend containing the OAUTH token authenticated the user, his GPS data and his selected vehicle's ID. It then starts an interval which is triggered every 5 second and send a new message to the server, with the updated GPS informations of the user. The response of this message, from the backend to the application, contains a "Match" field that can either be empty if not match is found, or filled with the informations of the spot and car the buyer was matched with. If this field is not empty, meaning a possible trade has been found, a new modal view is displayed to the buyer, containing the information of the spot and asking for confirmation.

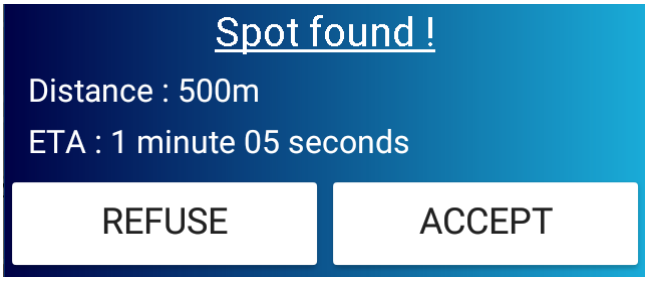


FIGURE 8.4: Spot found modal

The buyer can then either refuses the spot, which puts him back on search, or accept it, which initiates the trade. If he does, the location of the parking spot is registered internally by the application and displayed on the map. During his trip toward the spot, the interval keeps triggering, updating his GPS data in the backend, in order for the seller to see his spot's buyer location. During this phase, the two buttons becomes "Cancel" and

"Validate".

If he breaks the trade, by using the corresponding button, he faces the consequences we explained earlier, a loss of points (but a gain in reputation for being honest), the application then send a message to the backend indicating that the exchange has been broken, and the screen goes back to its default state.

The "Validate" button serve to notify that a trade a ended, and triggers a modal view allowing the user to give feedback on the spot exchange, which is send to the backend. The application then returns to its default state.

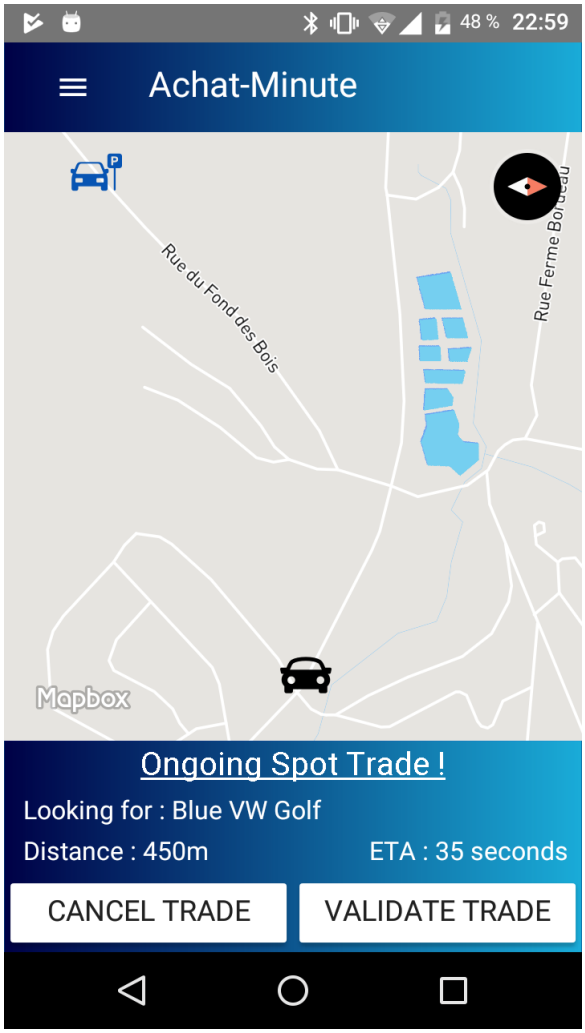


FIGURE 8.5: Screen during a trade

Selling a spot

If the user is selling his spot, the process behind it does not differ much. As soon as the confirmation is made by the user (hitting the "Confirm" button of the modal containing the sale's parameter), a message is send to the backend, containing the GPS datas of the spot, the time limit of the trade, and the vehicle's ID. Once again all messages are authentified thanks to an OAUTH token. An interval is also initiated, pinging the server every 5 seconds, updating GPS informations even though those should not be varying much, and similarly to the buying process, receives in the response either a match has been found or not. If no match is found during the time limit specified by the user, the trade offer is simply canceled, the user receives his StreetCreds and the map screen goes back to default.

If a match is found however, the seller gets notified and the location and description of the buyer's vehicle appears on his screen. He does not get the option to refuse the buyer. At that point, the buttons becomes "Cancel" and "Validate", and the process is nearly identical to the buying process. The applications keeps pinging the server, and receives in response, amongst other information about the trade, the location of the buyer, in order to update his position on the map for the seller, and also update his estimated time of arrival.

The "Cancel" and "Validate" buttons serves the same purpose as for a buyer. Once the trade is over, the screen goes back to its default state.

8.3.2 Login and Register

Those two screen serves the purpose expected of registration and login screens, without fanciness. The new account data are sent to the backend through simple POST request. If the registration is successful, the user is redirected to the login page, otherwise the registration page is reloaded and indicates the invalid fields.

As for the login screen, the only notable action other than the login itself his that the informations about the user's vehicle are fetch from the database once the login is confirmed and stored locally, in order to be avalaible immediately from the application without database access for the other pages and modal views of the application.

8.3.3 Options Screen

The options screen are splitted in 3 major pages : the vehicles page, the profile page, and the account management page.

Vehicle page

This page allows for the user to manage the vehicle he owns. It is presented in the form of a list containing the different vehicles the user has registered to his account, the list being obviously empty for new accounts. The button “Register” opens a modal view in which the user can indicate the informations about his car he agrees to show while trading spot. The license plate is required to register the car, but its visibility can be set to “Hidden”, meaning that it won’t show during a trade.

The list of car’s brand is generated from the database. Once the user select a brand, the model’s list become active and is filled with the differents models for the selected brand. This automation allows us to categorize the cars by their size, without having the user the estimate it himself, which in turn we use to ensure that people’s with bigger cars are not match with spot too small for their vehicle. The remaining fields (year of manufacture, description and color) are freely filled by the user depending on the informations he agrees to give.

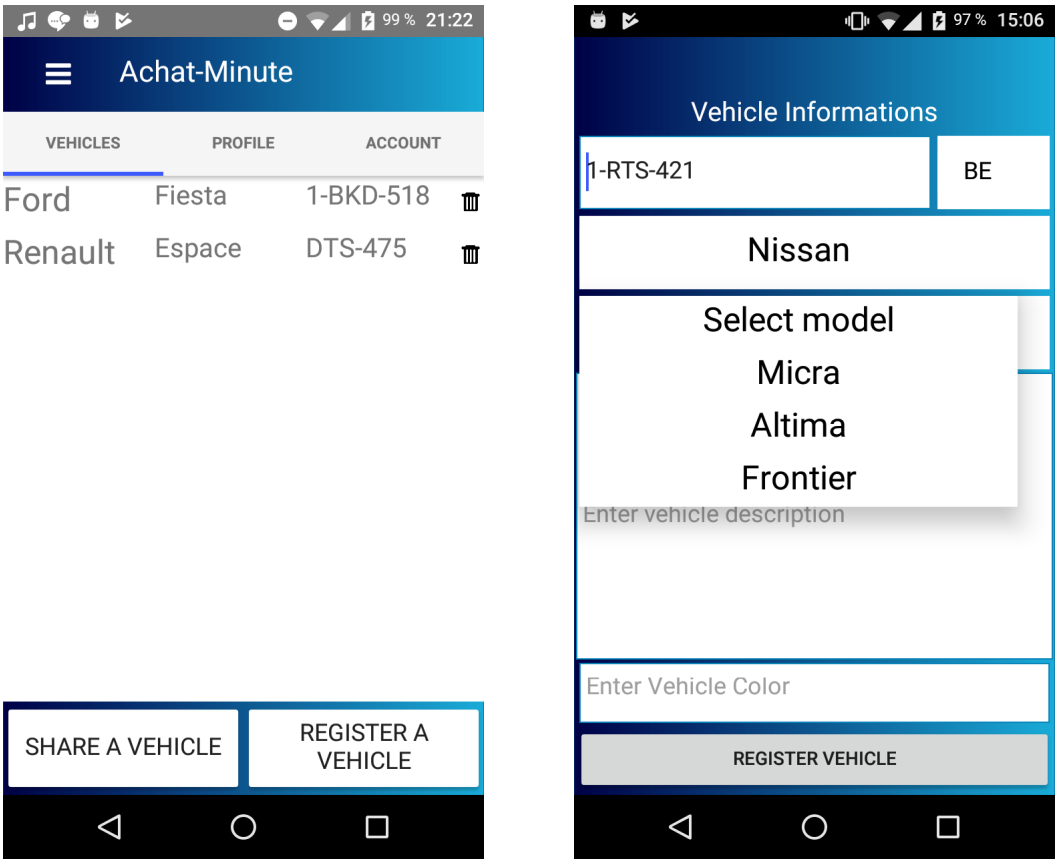


FIGURE 8.6: Vehicles list and details

Once the user validates the informations (by pressing the "Register Vehicle" button of the modal view), a POST request containing the vehicle's data is send to the server. If it is the first occurrence of the vehicle in the database (identified by the license plate), the vehicle is registered and linked to the user as the owner of the vehicle, and himself is redirect to the option page containing his vehicles list, where he now can see at least the vehicle he just added. In this page, by selecting a vehicle on the list, he opens the same modal view used for the vehicle registration, but this time with the fields pre-filled with the cars data, allowing him to modify it if he made a mistake, wants to add or remove some informations or edit the visibility of certain fields.

After the registration, the user can also send an invitation link for each car he owns (being the owner of a vehicle is defined by being the first person to register said vehicle in our database), allowing the invited users, identified by their email address, to also use the car. These users are notified when they log in for the first time after receiving the invitation, and if they accept, the vehicle also appears in their vehicle list, but they are not considered the owner and thus cannot share it with others. This system allows for a vehicle to be share between users, for example in the case of a family.

However, if during the registration, the vehicle already exists in our database, a pop-up window opens indicating that the vehicle has already being registered by another user, and that it is up to the owner to send the link allowing for the usage of the indicated vehicle. If the user is the real owner of the car and is the victim of prank or troll by another user that registered his car for him, he can then contact the support to restore his right to use the vehicle, if he is able to prove his ownership via legal documents.

8.3.4 Profile page

This page gives access to the user profile data, namely his current StreetCreds balance and a button allowing him to offer some of his StreetCreds to another user, an access to his trade history, and also some more technical informations, the display language of the application (the default language being the language of his smartphone operating system) and the option to activate or disable sound and notifications.

Trading StreetCreds only requires the email address of the receiver, and obviously having a positive amount of Creds, as there is some rare cases where a user account could be negative, (through breaking trades), and obviously, the user cannot gives more than he has. The gift take the form of a POST request to the server containing the receiver's data and the amount offered. The receiver's StreetCreds are then updated accordingly.

8.3.5 Account page

This page provides the standard services for a account management page, and allows for the user to update his account information, and modify his password.

8.4 Possible Improvements

8.4.1 Occupancy and reporting

At multiples occasions, we spoke about one of the main improvement to the application. The very first practical task we accomplish for CommuniThings was a small app designed to log and then save in CSV files the data of several sensors. The purpose of this application was to collect enough sensor's data to then be able to generate a decision tree, in order to determine if someone was walking, standing still, driving, or even using public transportations such a train, depending on the smartphones data.

This decision tree was done by a developer at CommuniThings, and one of the proposed usage of the tree was to implement it within a NativeScript module or a Java library, that we could then use within our own application. With such process in the application, we would be able to determine the status of the user, as long as he allows us to use the sensors. With those data, we could then determine when an user is leaving a parking spot, for example when his data indicates WALK-STAND-DRIVE on a short period of time, creating the Ephemeral parking spot we mentioned previously. This is a similar idea to what ParkSense has developed [25], that we presented in the first part.

But this remains a small scale solution to parking requests, only providing a marginal amount of parking spots, with a low chance of availability. A better solution would also to determine the parking events, when someone parks his car (typically DRIVE-STAND-WALK sequence of status), and on parallel obtain an estimation of the amount of places available in a specific geographic area (city district, large street, known parking lot). By having both the amount of places available and the park-unpark events, we could then produce an estimation of the occupancy of the area, and we could display this information on the map to redirect our users towards the place with the lowest occupancy close to them, improving their chances to find a spot. This idea was explored by a team of the University of British Columbia [7]. Displaying only the occupancy of the street and not the precise spots also help against the *multiple-car-chasing-single-space* syndrome, a common issue with in-spot sensors based SmartParking solutions.

The biggest issue for such system to work is that it requires a lot of people using the application, as the more parking events are detected, the better the estimation of the occupancy become. This means that such feature is virtually unusable at the launch of a

product such as our ParkExchange application. This feature is also vulnerable to external events lowering the amount of places available in an area, for example due to public works, flawing our estimation. This could however be prevented through user's reporting, for example using the model Waze[1] has developed. This would allow our users to send a message or a picture indicating that some actual event is reducing available spots quantity in this specific area. By increasing the precision of the information, our estimation of the occupancy in the area would then become more and more accurate leaving the error margin of spots used by normal drivers outside the community.

The large scale requirement was however to much of a constraint to really be meaningful to implement the system in the base version we developed. But should such application becomes popular, at least within a specific city, such system would definitely provide a huge boost of attractiveness to the application.

8.4.2 Dynamic Pricing

This is something we describe in the field review and thus we won't re-explain it extensively. Uber's model of pricing contains interesting elements that could prove useful to boost the willingness of our user base to share a spot. In our case, the geographical parameter would be a big part in a spot value, thus not only having a surge system in order to respond to increases of demand during peak hours, but to also respond to demands in more specific, well-known or busy areas through a parallel system should be consider, similar to what SFpark [29] is achieving, but without the need for sensors.

One of the options would be to determine a spot's value compared to another simply by obtaining the average amount of buyer's in a close range of the spot. The more buyers in the area, the more this specific spot gains value, independently of other spots sold at the same time. We would then have two values multiplying a spot's cost and gain, a surge value similar to Uber to increases our general prices during peak hours, and that would be variating quickly, depending on the Supply/Demand of the previous minutes, and another value being less volatile and calculated day after day to determine the popularity of certain areas in term of parking, and adjusting the prices of the parking spots in said areas accordingly.

Once again however, such system is hard to put in motion without large amount of data and a large user-base already established. This is why we focused on a simple, more straightforward static economic system in our prototype.

[1] https://www.waze.com/

8.4.3 Friends

A recurring feature in any application with a social aspect, which our certainly has, is the Friends List. Either by importing your contacts, or by adding people in it manually, having a list of people you know and trust, that also use the application is beneficial for the community as a whole.

We pointed this in the Community chapter, users are more willing to share a spot with users they know, and already knowing members of the community increases the probability that another user joins it. The friends list could then serve two purpose :

- Encouraging the recruitment of a contact, eventually gaining bonuses at the same time
- Encouraging sharing a spot when friends are looking for one nearby

The former could be similar in what World of WarCraft, that we mentioned before, does to expand its community. The basic idea behind the "Recruit-a-friend" [37] program is that you can send an invitation to a person you know that is not yet a member of the community (based on email message). If they join and becomes active members, you and your friend gains bonuses. In our case, this could lead to free "StreetCreds", and reduced spot costs for a month. This program, in conjunction with the claim established before that members join a community more easily when they know someone already inside, should increase our user-base.

The second idea is the more practical one in order to increase the amount of spots shared through the application. Once again, this is in accordance with a claim made previously, peoples are more willing to help users they know well. Thus designing a notification indicating to users when one of their friends is looking for a parking spot nearby, and that should they share their spot, their friend would automatically receives it, it should improves their willingness to trade a spot, knowing that it would benefit a friend immediately.

8.4.4 Buying a spot in advance

Contrary to the seller's ability to plan his offer in advance by setting a buffer time that delays the exchange, the buyer has no similar capability. When a user press the button to buy a spot, the system assumes that the buyer is looking for a spot near his current location. It would be interesting to allow the buyer to request a spot at a remote location in the future, or even allow a user to sell the spot he currently occupies while simultaneously requesting a spot at a different location in the future. This possibility could further incentivize users to shop in downtown area, where it is often needed to take short trips between shops, by allowing them to plan their next stop more easily. It would also open up new opportunities to incentivize users to participate more actively, perhaps by prioritizing parking requests coming from users who are also selling a spot at that same time.

However, the implementation of this capability would require the reworking of the exchange flow and associated assumptions. Furthermore, the location of the parking request would need to be logically separated from the buyer's current location.

8.4.5 Improved communications between buyers and sellers

Buyer stuck in traffic, seller having a sudden emergency and having to know precisely when the buyer will arrive, lack of communications during a trade, those are issues that could lead to failed or contested trade. In order to solve some of those, Originally we wanted to use VoxBone [2] within our application. Due to a lack of time we ultimately decided that it was not a priority but this remains a possible improvement of the application.

VoxBone offers a VoIP service, alongside DID [3] numbers. With it, we could have implemented an option to call or send a message the other party of a trade, without using your real phone number, thus keeping anonymity and personal information safe. With such system in place, should an issue arise during a trade, the affected user could inform the other party and possibly negociate a solution that satisfies both parties.

[2] https://www.voxbone.com/about-voxbone/overview
[3] https://www.voxbone.com/services/did-numbers

Conclusion

Mobility, pollution and urbanization have been three major concerns for cities since many years, and the constant increase in the amount of vehicle contributes to the worsening of the situation. It is thus logical that with the apparition of SmartCities projects, the parking issue in downtown areas would be the focus of a lot of research. Many smartparkings systems have been developed in the recent years, but it is clear that this technology is still in its infancy and large-scale deployment is not yet achieved.

From SPARK reliance on sensors, to Open Spot lack of incentive, those solutions would have problems to cope with large scale, on-street usage. But it does not mean they are without merits. The reservation method for example, used in several prototype, improved greatly the probability of a user getting a parking spot faster, compared to other method that often lead multiple drivers to the same spot, a recurrent issue with Parking Guidance Information systems. Open Spot also opened the way of the usage of public parking spot without the use of sensors, making the system considerably cheaper than systems relying on extensive infrastructure such as SFpark.

Taking into account crowdsourced possibilities, and making people interacting directly with one another while looking for a parking spot is a direction that is not commonly explored in research. Although, such model could benefit from the same advantages as reservation-based systems, but without the need of dedicated sensors, as the parking informations are provided through crowdsourcing. It is not without a cost however. Using crowdsourced information and relying on people selling spots and waiting for strangers to come and take their parking spot requires a lot of consideration towards the inner mechanisms of online communities, and what make people willing to be part of them.

Indeed, a crowdsourced application with a too small community is doomed to fail, but options exists to create such a community. The parking facilities provided by the application is its most attractive incentive, and in itself could already attract a lot of members. Creating a sense of community, that improves the quality of life in the user's city, is also a direction to consider when designing crowdsourced systems. And of course, a factor to consider when talking about crowdsourcing is the friends and related impact on the willingness of others to be part of a community. Implementing tools that give an importance to those relations is a safe way to encourage mass adoption.

Another aspect that is usually not considered by smartparking systems, and rightfully so, is the development of a virtual economy alongside the community. But with complete end-to-end trade of parking spot, it has to be taken into account. *How much does buying a spot cost ? How much do I gain by selling my spot ?* Those questions have to be resolved for a reservation and trade based system to work. On that matter, dynamic pricing which take into account the location of the spot and the time of day are most likely the way to go for crowdsourcing models, as more rewarding trades should lead to more offers.

And of course, when talking about mobile application, security and privacy should always be a concern, even more when the system has to store personal information, which the exchange-based application we presented has to do. Particularly, the system has to know the position of both users during the entire duration of the exchange, as well as sensitive information on their identity and vehicle. We thus reviewed the most common security risks in the mobile application space and identified mitigation strategies. We also explored the impact privacy concerns may have on user satisfaction.

All those themes are important to consider for the development of a crowdsourced smartparking application. Such system still has to go mainstream, but it is more than likely coming. Alongside smarticities project, solutions to parking issues will keep getting developed, and crowdsourcing is certainly an interesting direction to try out.

The prototype developed during this thesis can be seen as a first iteration of a crowdsourced smartparking application, embedded in the StopBuy application developed by CommuniThings. The complete exchange process is implemented, allowing users to offer and buy spots. Some elements are not implemented however, mostly related to the community aspect of the application, and are the main future improvements that would change the prototype into a real application.

The "Crowdsourced smartparking" model presented in this thesis is inspired in large parts by reservation-based systems but crucially, it solves the important issue of the guaranteed availability of the reserved spot. However, there are a lot of opportunities to improve the practicality of such system, and to ensure its future. As pointed out several time, an issue with crowdsourcing and social applications as a whole is to reach a critical mass of users using the system and participating in it. The options presented to increase commitment and the userbase are a start, but questions about sustainability remain, especially at launch. Research on techniques to recruit the very first users and how to make the service sustainable even with a low amount of user could be an interesting follow up to this thesis.

Another aspect worth looking into would be the usage of privately owned parking

spots that the owners could offer through the system when they are not using it, for example the space in front of a garage door on a street. It could be an option to increase the amount of spots available to the community, but also causes additional concerns such as ensuring that the buyer does leave the spot in time.

Finally, another direction that such application could take in the future would be to be merged with other driving related applications and system such a GPS guidance. For example, having a GPS and parking application combined that would recommend small variation in the user's destination to redirect him nearby people offering spots, eventually start the exchange process automatically, and finally guide the driver directly to the spot trade's location. Such a combination could be a good way to attract more user, and to make the application adopted by a large amount of users quickly.

Even though it is not up to us whether our prototype is further developed some day, or if some component of the system can be reused, it was an exciting journey. In the future, we will keep looking at emerging Smartparking solutions as they become more and more mainstream, with the hope that we can, some day, identify some system that contains concepts we explored and be able to tell ourselves that we were going in the right direction. In any case, this thesis has been a huge opportunity for us to discover and explore modern application of information technologies that were truly new to us, and will without a doubt have an important impact on our future careers.

Gary BOGAERTS & François JONES

Glossary

Buffer Time the Buffer Time is the period during which the seller is not ready to leave immediately during an offer.

Parking Buyer a user is referred to as a Parking Buyer or simply buyer when this user has an active parking request registered.

Parking Seller a user is referred to as a Parking Seller or simply buyer when this user has an active parking request registered.

StreetCred StreetCreds are the currency used to trade parking spots inside the ParkExchange system.

Time of sale The Time of sale is the sum of Buffer Time and Wait Time and is equal to the total duration of the offer.

Wait Time the Wait Time is the period during which the seller is ready to realize an exchange with a buyer at a moment's notice.

Bibliography

[1] *About NativeScript*. https://www.nativescript.org/about. Accessed: 2017-08-13.

[2] Richard Arnott, Tilmann Rave, Ronnie Schöb, et al. "Alleviating urban traffic congestion". In: *MIT Press Books* 1 (2005).

[3] Lars Backstrom et al. "Group formation in large social networks: membership, growth, and evolution". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 44–54.

[4] *Banished to prisoner's island*. http://nexus.leagueoflegends.com/2017/01/ask-riot-banished-to-prisoners-island/. Accessed: 2017-08-13.

[5] Susan L Bryant, Andrea Forte, and Amy Bruckman. "Becoming Wikipedian: transformation of participation in a collaborative online encyclopedia". In: *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*. ACM. 2005, pp. 1–10.

[6] M Keith Chen and Michael Sheldon. "Dynamic Pricing in a Labor Market: Surge Pricing and Flexible Work on the Uber Platform." In: *EC*. 2016, p. 455.

[7] Xiao Chen, Elizeu Santos-Neto, and Matei Ripeanu. "Crowdsourcing for on-street smart parking". In: *Proceedings of the second ACM international symposium on Design and analysis of intelligent vehicular networks and applications*. ACM. 2012, pp. 1–8.

[8] Yan Chen et al. "Social comparisons and contributions to online communities: A field experiment on movielens". In: *The American economic review* 100.4 (2010), pp. 1358–1398.

[9] AL Crain, AM Omoto, and M Snyder. "What if you can't always get what you want? Testing a functional approach to volunteerism". In: *annual meetings of the Midwestern Psychological Association, Chicago, IL*. 1998.

[10] B. Dumas. *INFOM450: Internet of Things : 1 – Introduction*. Slides for the INFOM450 courses. Feb. 2017.

[11] Jerry Gao et al. "Mobile Testing-as-a-Service (MTaaS)–Infrastructures, Issues, Solutions and Needs". In: *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*. IEEE. 2014, pp. 158–167.

[12] Yanfeng Geng and Christos G Cassandras. "New "smart parking" system based on resource allocation and reservations". In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1129–1139.

[13] Jonathan Hall, Cory Kendrick, and Chris Nosko. "The effects of Uber's surge pricing: A case study". In: *The University of Chicago Booth School of Business* (2015).

[14] MYI Idris et al. *Car park system: a review of smart parking system and its technology*. 2009.

[15] Nicolas Kaufmann, Thimo Schulze, and Daniel Veit. "More than fun and money. Worker Motivation in Crowdsourcing-A Study on Mechanical Turk." In: *AMCIS*. Vol. 11. 2011. 2011, pp. 1–11.

[16] J Kincaid. *Googles open spot makes parking a breeze, assuming everyone turns into a good samaritan*. 2010.

[17] Jennifer King. "How Come I'm Allowing Strangers to Go Through My Phone? Smartphones and Privacy Expectations." In: (2012).

[18] Robert E Kraut et al. *Building successful online communities: Evidence-based social design*. Mit Press, 2012.

[19] *KurbKarma: A Social Network, And App, To Find Parking Where And When You Need It*. https://techcrunch.com/2012/05/21/kurbkarma-a-social-network-and-app-to-find-parking-where-and-when-you-need-it/. Accessed: 2017-08-13.

[20] Hong Lu et al. "The Jigsaw continuous sensing engine for mobile phone applications". In: *Proceedings of the 8th ACM conference on embedded networked sensor systems*. ACM. 2010, pp. 71–84.

[21] Nicolas Michinov, Estelle Michinov, and Marie-Christine Toczek-Capelle. "Social identity, group processes, and performance in synchronous computer-mediated communication." In: *Group Dynamics: Theory, Research, and Practice* 8.1 (2004), p. 27.

[22] Henry Muccini, Antonio Di Francesco, and Patrizio Esposito. "Software testing of mobile applications: Challenges and future research directions". In: *Proceedings of the 7th International Workshop on Automation of Software Test*. IEEE Press. 2012, pp. 29–35.

[23] Logeshwaran Murugesan and Prakash Balasubramanian. "Cloud based mobile application testing". In: *Computer and Information Science (ICIS), 2014 IEEE/ACIS 13th International Conference on*. IEEE. 2014, pp. 287–289.

[24] Anandatirtha Nandugudi et al. "Pocketparker: Pocketsourcing parking lot availability". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM. 2014, pp. 963–973.

[25] Sarfraz Nawaz, Christos Efstratiou, and Cecilia Mascolo. "Parksense: A smartphone based sensing system for on-street parking". In: *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM. 2013, pp. 75–86.

[26] Peter Organisciak. *Motivation of Crowds: The Incentives That Make Crowdsourcing Work «Crowdstorming*. 2008.

[27] *OWASP homepage*. https://www.owasp.org/index.php/OWASP_Mobile_Security_Project. Accessed: 2017-08-13.

[28] *ParkTag | Free Social Parking App*. http://parktag.mobi/. Accessed: 2017-08-19.

[29] Gregory Pierce and Donald Shoup. "Sfpark: Pricing parking by demand". In: *Access Magazine* (2013).

[30] I Sherwin. "Google Labs' open spot: A useful application that no one uses". In: *URL: http://www.androidauthority.com/google-labs-open-spot-a-useful-application-that-no-one-uses-15186/, Visited in August* (2014).

[31] Irina Shklovski et al. "Leakiness and creepiness in app space: Perceptions of privacy and mobile app use". In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM. 2014, pp. 2347–2356.

[32] Donald C Shoup et al. *The high cost of free parking*. Vol. 206. Planners Press Chicago, 2005.

[33] SV Srikanth et al. "Design and implementation of a prototype smart PARKing (SPARK) system using wireless sensor networks". In: *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*. IEEE. 2009, pp. 401–406.

[34] Arvind Thiagarajan et al. "Cooperative transit tracking using smart-phones". In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2010, pp. 85–98.

[35] Luis Von Ahn. "Games with a purpose". In: *Computer* 39.6 (2006), pp. 92–94.

[36] Hongwei Wang and Wenbo He. "A reservation-based smart parking system". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*. IEEE. 2011, pp. 690–695.

[37] *World of Wacraft - Recruit a Friend*. http://eu.battle.net/wow/en/game/recruit-a-friend/. Accessed: 2017-08-13.

[38] Tingxin Yan et al. "CrowdPark: A crowdsourcing-based parking reservation system for mobile phones". In: *University of Massachusetts at Amherst Tech. Report* (2011).